**DS 620 Machine Learning and Deep Learning**

**HOS09Aa: Computer Vision with CNN in TensorFlow**

05/19/2022 Developed by Chunhui Xu

05/21/2024 Reviewed by Naveena Moddu

08/21/2025 Reviewed by Ashwin Paturi

10/30/2025 Reviewed by Kina Anh Bui

School of Technology & Computing (STC) @ City University of Seattle (CityU)

A picture containing text, clipart

Description automatically generatedA picture containing icon

Description automatically generated

**Before You Start**

* The directory path shown in screenshots may be different from yours.
* The document uses Google Colaboratory as the default compiler. If you want to run the code on a local machine, you need to configure the environment on your own.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning objectives**

* Convolutional Layer
* Filters
* CNN and VGG

**Resources**

* deeplizard. (2017, December 9). *Convolutional Neural Networks (CNNs) explained*. <https://deeplizard.com/learn/video/YRhxdVk_sIs>
* Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow* (2nd ed.). O’Reilly Media.
* Gupta, S., Arbelaez, P., & Malik, J. (2013). Perceptual Organization and Recognition of Indoor Scenes from RGB-D Images. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 564–571. <https://doi.org/10.1109/cvpr.2013.79>
* John. (2015, February 5). *How to detect texture or non-texture in image* [Stack Overflow Question]. Stack Overflow. <https://stackoverflow.com/questions/28347547/how-to-detect-texture-or-non-texture-in-image>
* Krizhevsky, A. (2009). *The CIFAR-10 dataset*. Computer Science University of Toronto. <https://www.cs.toronto.edu/%7Ekriz/cifar.html>
* Lendave, V. (2021, June 18). *What Is a Convolutional Layer?* Analytics India Magazine. <https://analyticsindiamag.com/what-is-a-convolutional-layer/>
* Ringwald, D. (2019, March 12). *Edge Detection in Opencv 4.0, A 15 Minute Tutorial*. Medium. <https://medium.com/sicara/opencv-edge-detection-tutorial-7c3303f10788>
* Sharma, N. (2019, December 18). *Introduction to basic object detection algorithms.* Medium. <https://heartbeat.comet.ml/introduction-to-basic-object-detection-algorithms-b77295a95a63>

**Introduction:** How do machines perceive an image? How can they distinguish between different elements of an image or between other animals, such as cats and dogs? The Convolutional Neural Network is the number one choice as it is best known for its application in analyzing images for computer vision tasks. In this exercise, we will explore the components of a Convolutional Neural Network and its effectiveness in detecting patterns in images.

**Dataset:** The dataset we will use in this HOS is known as CIFAR-10. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

手机屏幕截图

中度可信度描述已自动生成

Figure 1: CIFAR-10 Dataset (Krizhevsky, 2009)

More details on the dataset can be found here: <https://www.cs.toronto.edu/~kriz/cifar.html>.

**Convolutional Layer:** Before introducing the convolutional layer, it is helpful to review the fully connected layer. The fully connected layer, as the name implies, means that every node in the next layer is related to every node in the previous layer. Thus, each node in the next layer is an information optimization of the nodes in the entire previous layer. For example, assume the upper layer has three nodes, a, b, and c, and the lower layer has two nodes, x and y. At this point, x and y are features based on a, b, and c, with y being independent of x.

In this case, the Weight and Bias of this fully connected layer is:

If the input is a 32\*32\*3 image and the output is a fully connected layer of ten nodes, then each output node is a new feature integrated from each input pixel. However, there is an issue. If we intend to tell what the object in the picture is (for example, a cat or a dog), we do not need to look at the whole picture. In fact, we only need to look at their tails, heads, and paws to tell them apart. In other words, we do not need to see the whole picture at all; just some of the information is enough. This differs from the fully connected layer, where each node includes all the information from the previous layer. Therefore, we need another structure that scans only part of the image, and the convolutional layer does that. For a fully connected layer with 3072 input nodes and 10 output nodes, the input is a 3072-dimensional matrix (Flattened from a 32x32x3 image), the weight matrix is 3072x10, and the bias vector is 1x10. Since we only need partial information from a 3\*3 area of the 32\*32 image, we can use a 3\*3 weight to connect each 3\*3 part of the image. We call the 3\*3 weight a kernel in a convolutional layer. In summary, when the input is a 32\*32\*3 image (the first 32\*32 is the size of the picture and the next 3 is the number of channels of the picture - the number of channels of color pictures is usually 3), the kernel size is 3\*3\*3 (the first 3\*3 corresponds to the size of the kernel scanned on the image and the next 3 indicates the number of channels of the input image). The following figure shows a single-channel 6\*6 image passing through a 3\*3 kernel to get a new 4\*4 image (in this example, the bias is 0).

图示

描述已自动生成

Figure 2: Convolution Operation (Lendave, 2021)

**Filters:** We can think of each kernel as a filter because each kernel is scanning the entire image to filter out new information. Only one new feature map can appear after each filtering. Only one feature map is entirely inadequate to meet our needs; after all, if we were to classify cats and dogs, only the information of their tails would not be enough. Therefore, we need additional filters to exclude more infographics and feature maps. If our input image is 32\*32\*3, the size of the convolution kernel is 3\*3, and the number of filters used is 16, then we will get a feature map of 30\*30\*16. If we want the input and output images to be the same size, we can use padding. After padding, the output size is 32\*32\*16.

The process of deriving from fully connected layers to convolutional layers is explained above. For better understanding, let us now reverse it. When the kernel size is equal to the input image size, the convolutional layer is equal to a fully connected layer. The filter number in the convolutional layer is the node number in the fully connected layer.

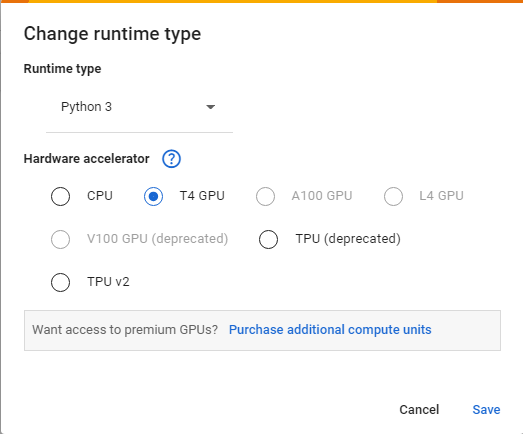
**Import Libraries**

1. Go to <https://colab.research.google.com>. Create a new notebook and name it “CNN\_tensorflow.ipynb”.
2. To speed up training, we will set the hardware accelerator. Go to Runtime > Change runtime type:

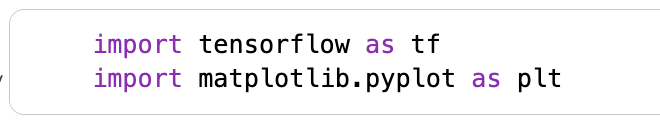
Graphical user interface, application, Word

Description automatically generated

1. Select GPU as the Hardware accelerator and press Save:

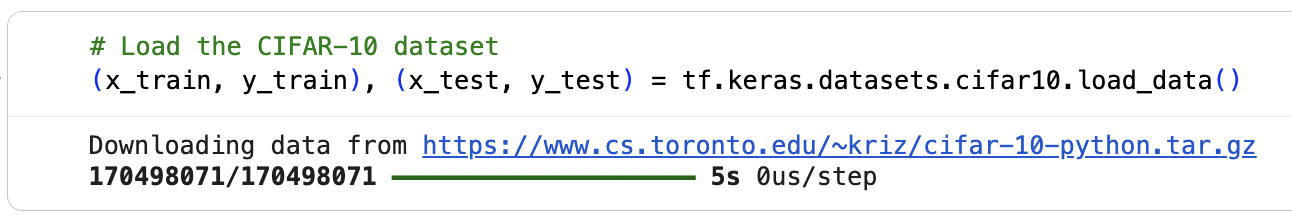


1. Now, we need to import the necessary libraries. Type the following in your code cell, then click the play button or Ctrl + Enter to run the cell.

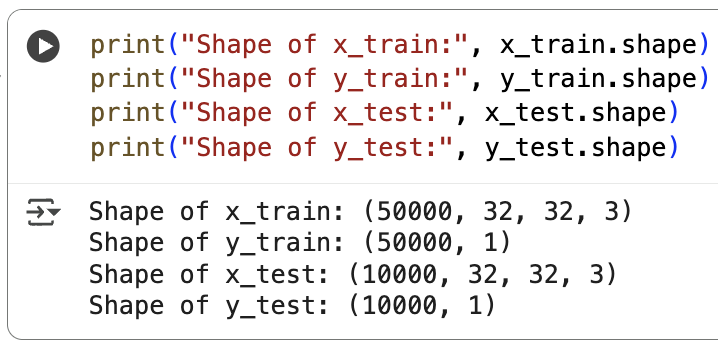


**Data Preparation**

1. As mentioned previously, we will use the CIFAR-10 dataset for this exercise. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.



1. Next, we will check the data size. Click the + Code button to add new code cells. Type the following code in the cells, then click the play button or Ctrl + Enter to run them.



Here we can see there are 50,000 train images and 10,000 test images. Each image size is 32\*32\*3. The labels of the images are numbers between 0-9 (ten possible labels), so the label’s shape is 1.

1. Now, let us check the first ten images in the training set. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.

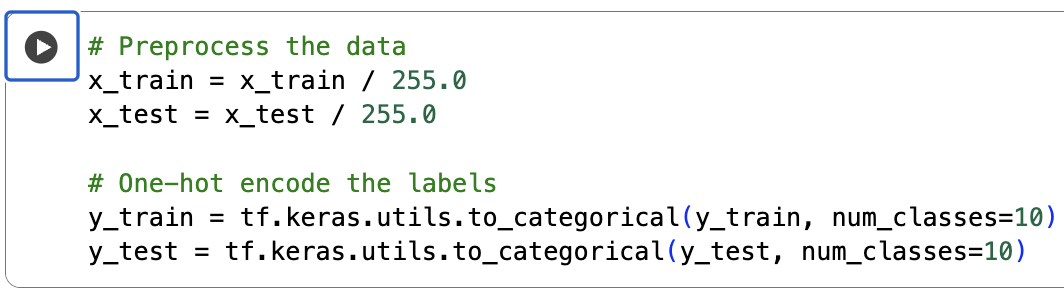


Each of these images is in RGB format, so each pixel has a value between 0 and 255. Also, their labels are all integers. Therefore, we need to preprocess them. For the image part, we can normalize by dividing by 255. For the labels, we can convert to tensors by one-hot encoding. The one-hot encoding is a helpful method in image classification. For this dataset, the label is from 0 to 9 (a total of 10 different numbers). After one hot encoding, these 10 numbers will be 10 tensors of length 10:

0 => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
1 => [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]  
2 => [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]  
3 => [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]  
4 => [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  
5 => [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]  
6 => [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]  
7 => [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]  
8 => [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]  
9 => [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

In this case, we can use a fully connected layer with 10 output nodes as the output layer of the classification model. After obtaining the model's result, we can use one-hot decoding to convert the output tensors to integers. One-hot decoding is used to return the index of the maximum value in a tensor (or a list).

1. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.



**Simple CNN**

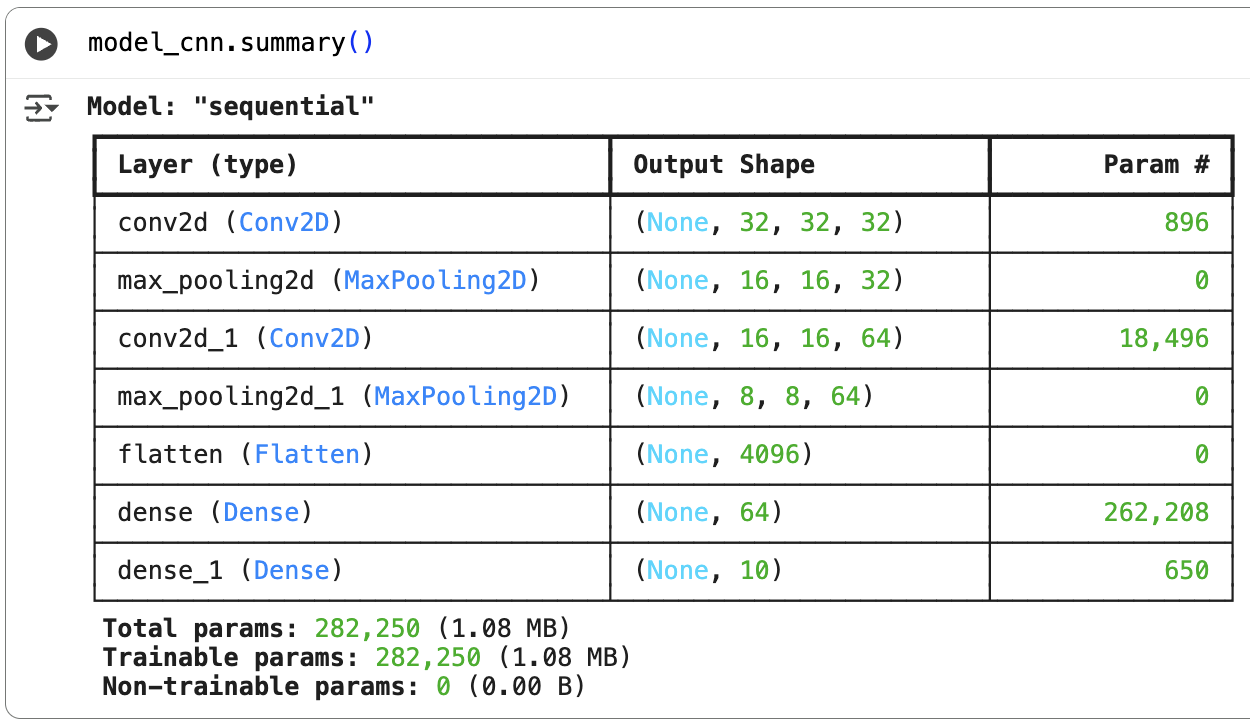
In this section, we will build a simple CNN using the Conv2D and MaxPooling2D layers. MaxPooling is a common operation in image processing to downsample. For example, if I use a MaxPooling layer with a pool size of (2, 2), this layer will use a (2, 2) size pooling kernel to scan the image, and only keep the maximum value. Therefore, if the input image size is 32\*32, the output size should be 16\*16.

1. Let’s create our simple CNN model. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.

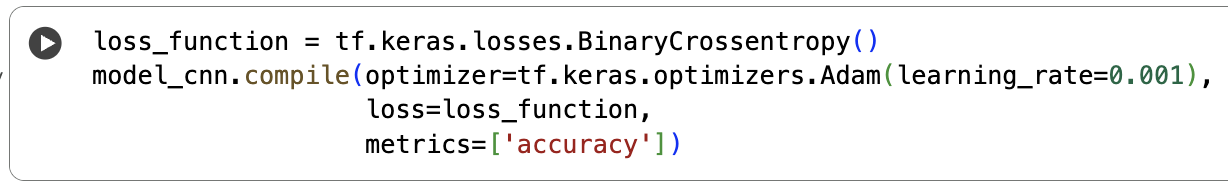


The output activation function is SoftMax, which is very useful in image classification. SoftMax can make each element of the output tensor positive and sum to 1. In other words, we can use the output of SoftMax as the probability of the image being classified into each type.

1. Before we compile our model, let us check the model details. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.

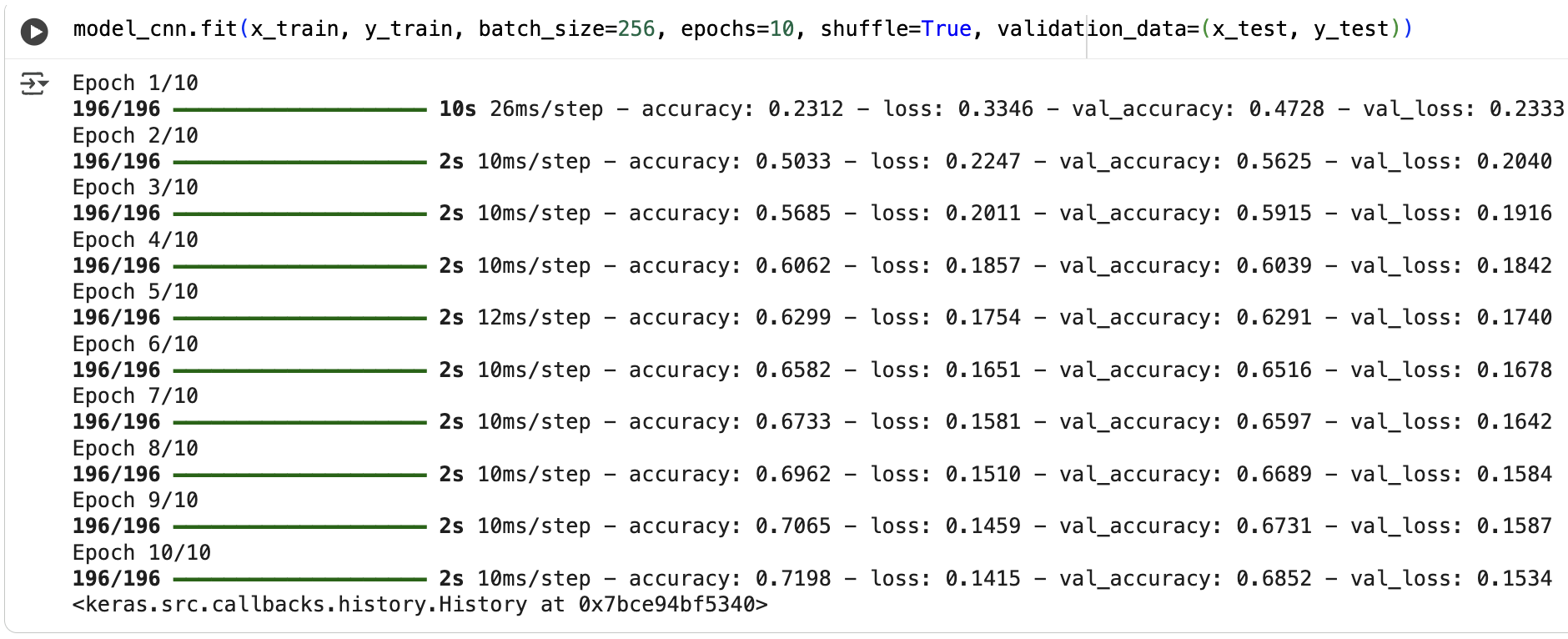


1. Now, we can compile the model. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.



The loss function in this model is cross-entropy, which is commonly used with SoftMax output. This loss function could help the model with SoftMax output be trained quickly.

1. Next, we will train our model. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.



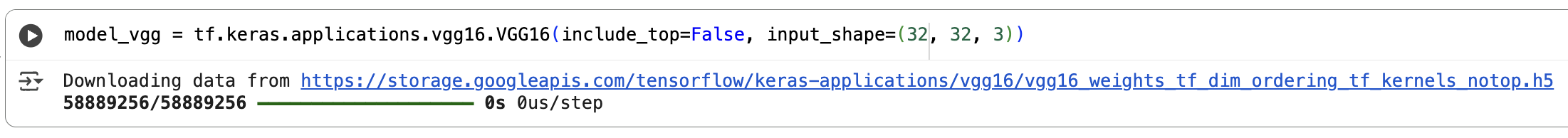
val\_loss and val\_accuracy are the loss and accuracy of the test set. When we evaluate accuracy, TensorFlow supports the use of one-hot decoding by default for model output.

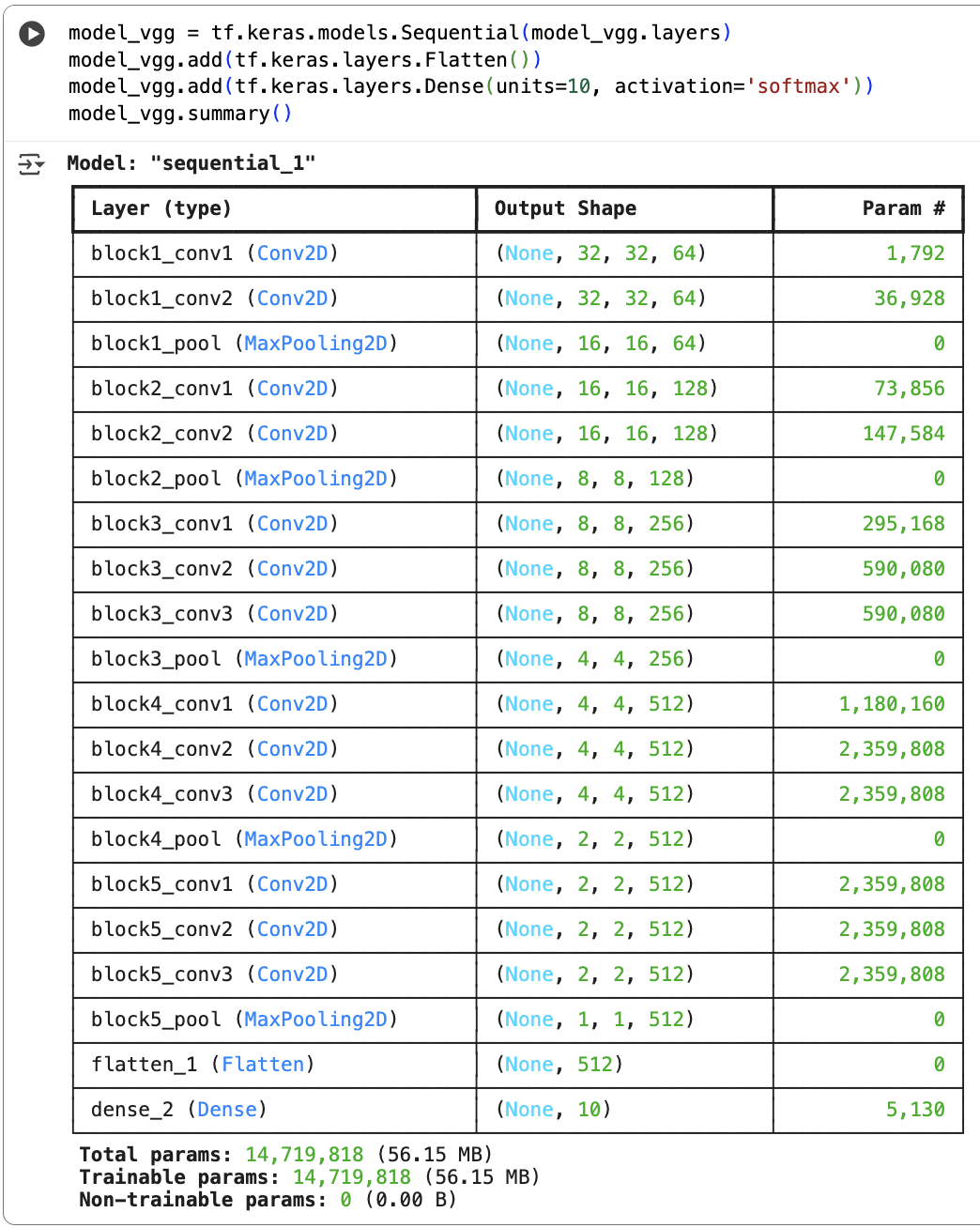
**VGG**

The VGG model is an already-architected model based on a CNN. It has demonstrated very powerful performance on image classification problems. We can load this model in many deep learning libraries and use it directly. We will use the VGG16 model in this HOS.

The VGG16 model is built for an image size of 224\*224\*3 and output classes of 1000. Therefore, we need to remove some parts of the original model and add some code to match our project.

1. We will need to load the VGG16 model, set the input size, output layer, and view its summary. Click the + Code button to add new code cells. Type the following code in the cells, then click the play button or Ctrl + Enter to run them.

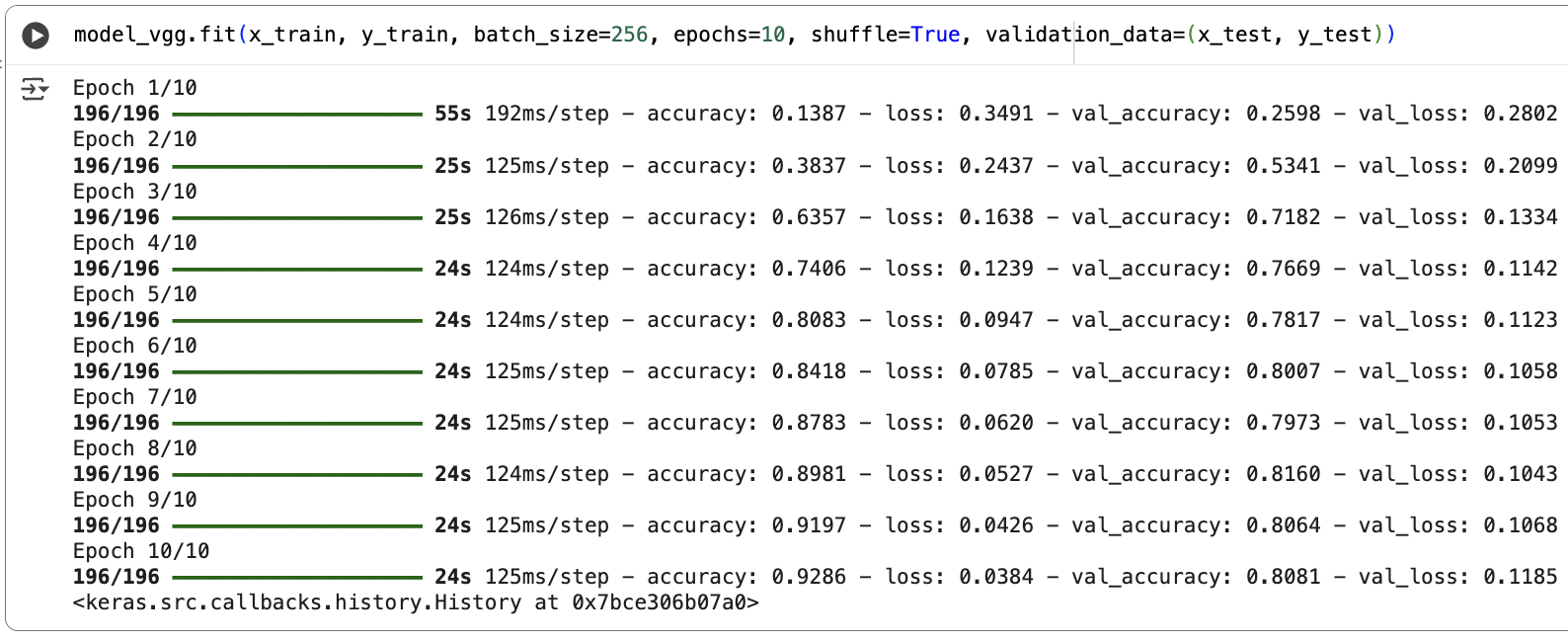




1. Next, let’s compile the model with the loss function we have previously defined. Click the + Code button to add a new code cell. Type the following code in the cell then click the play button or Ctrl + Enter to run the cell.



1. Finally, let’s train and evaluate the model. Click the + Code button to add a new code cell. Type the following code in the cell, then click the play button or Ctrl + Enter to run the cell.



**Push your work to GitHub**

Click File → Download → Download .ipynb.

The file will be downloaded to your Download folder (or another folder according to your browser settings).

Move the downloaded ipynb file to your Module09 folder.

Follow the instructions on the CityU STC TA Center Github.io [Submit your work page](https://cityuseattle.github.io/docs/hoporhos/submit/).