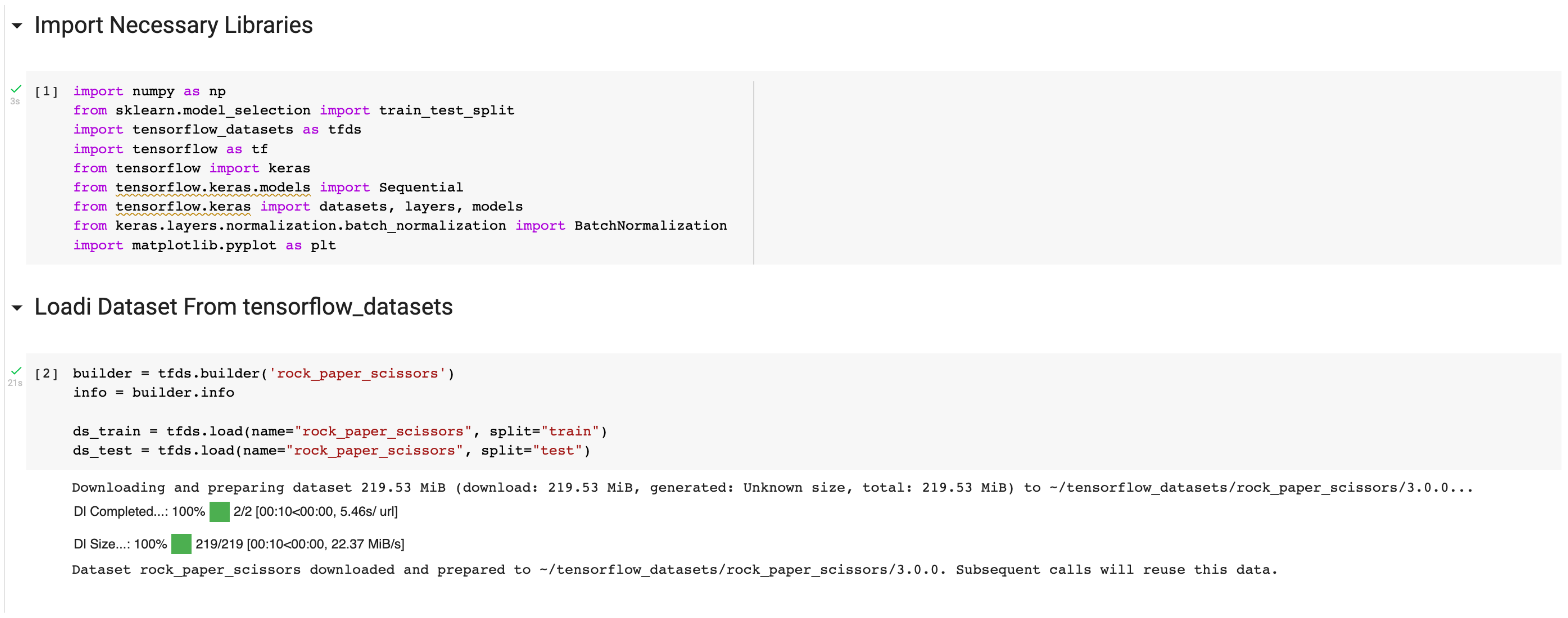
CNN-Rock Paper Scissors-Recognition

Kian Anvari Hamedani



Import Necessary Libraries

```
[1] import numpy as np
    from sklearn.model_selection import train_test_split
    import tensorflow_datasets as tfds
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.models import Sequential
    from tensorflow.keras import datasets, layers, models
    from keras.layers.normalization.batch_normalization import BatchNormalization
    import matplotlib.pyplot as plt
```

Loadi Dataset From tensorflow_datasets

```
[2] builder = tfds.builder('rock_paper_scissors')
    info = builder.info

    ds_train = tfds.load(name="rock_paper_scissors", split="train")
    ds_test = tfds.load(name="rock_paper_scissors", split="test")
```

```
Downloading and preparing dataset 219.53 MiB (download: 219.53 MiB, generated: Unknown size, total: 219.53 MiB) to ~/tensorflow_datasets/rock_paper_scissors/3.0.0...
Dl Completed...: 100% 🟩 2/2 [00:10<00:00, 5.46s/ url]

Dl Size...: 100% 🟩 219/219 [00:10<00:00, 22.37 MiB/s]
Dataset rock_paper_scissors downloaded and prepared to ~/tensorflow_datasets/rock_paper_scissors/3.0.0. Subsequent calls will reuse this data.
```

We import the required libraries for solving the question and training of neural network.

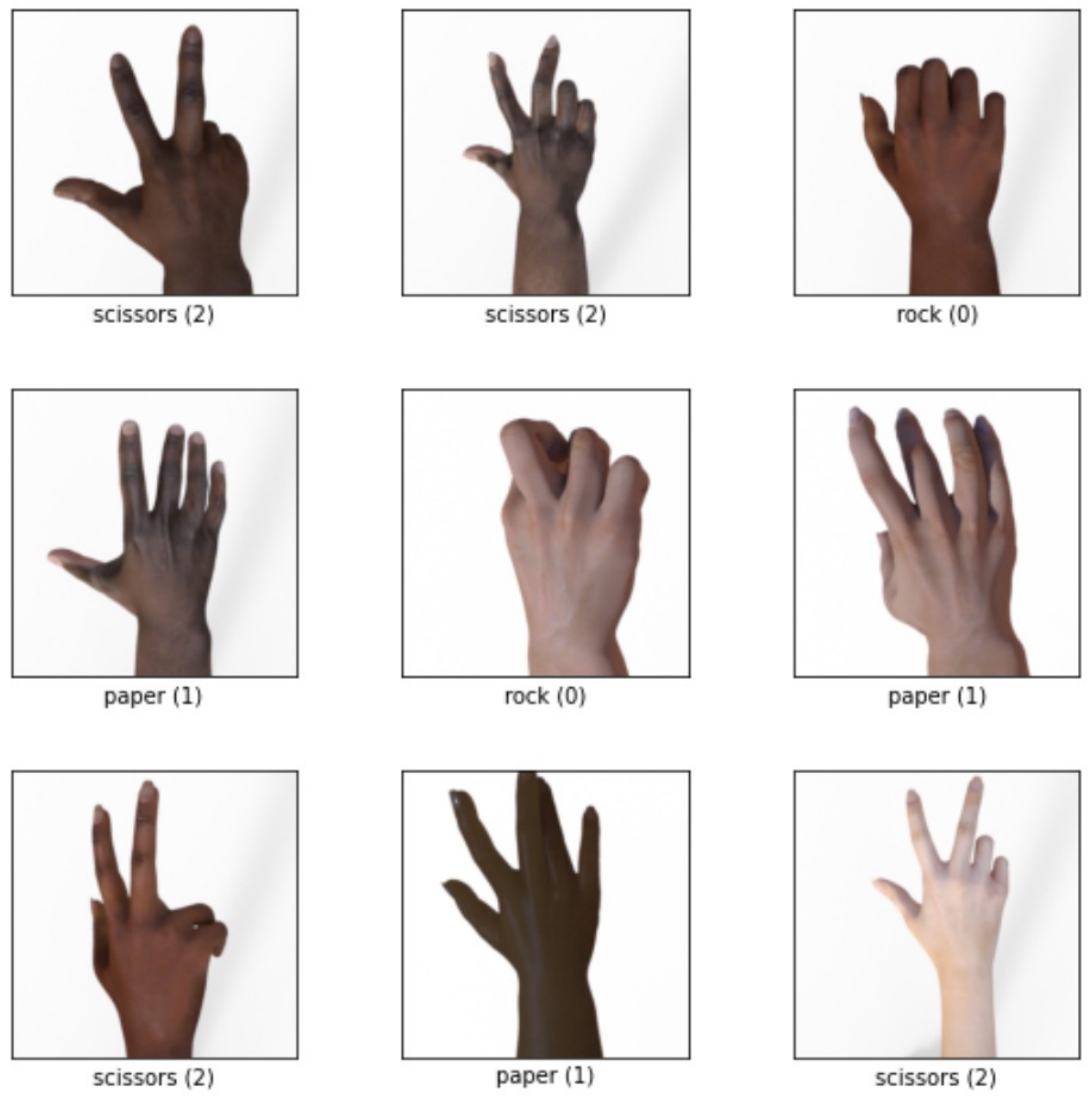
Using the Tensorflow_datasets library and the builder method, we load the desired dataset, i.e. rock_paper_scissors.

By default, this dataset considers part of the data as training data and another part as test data which we use.

Show Examples of 'rock_paper_scissors' Dataser

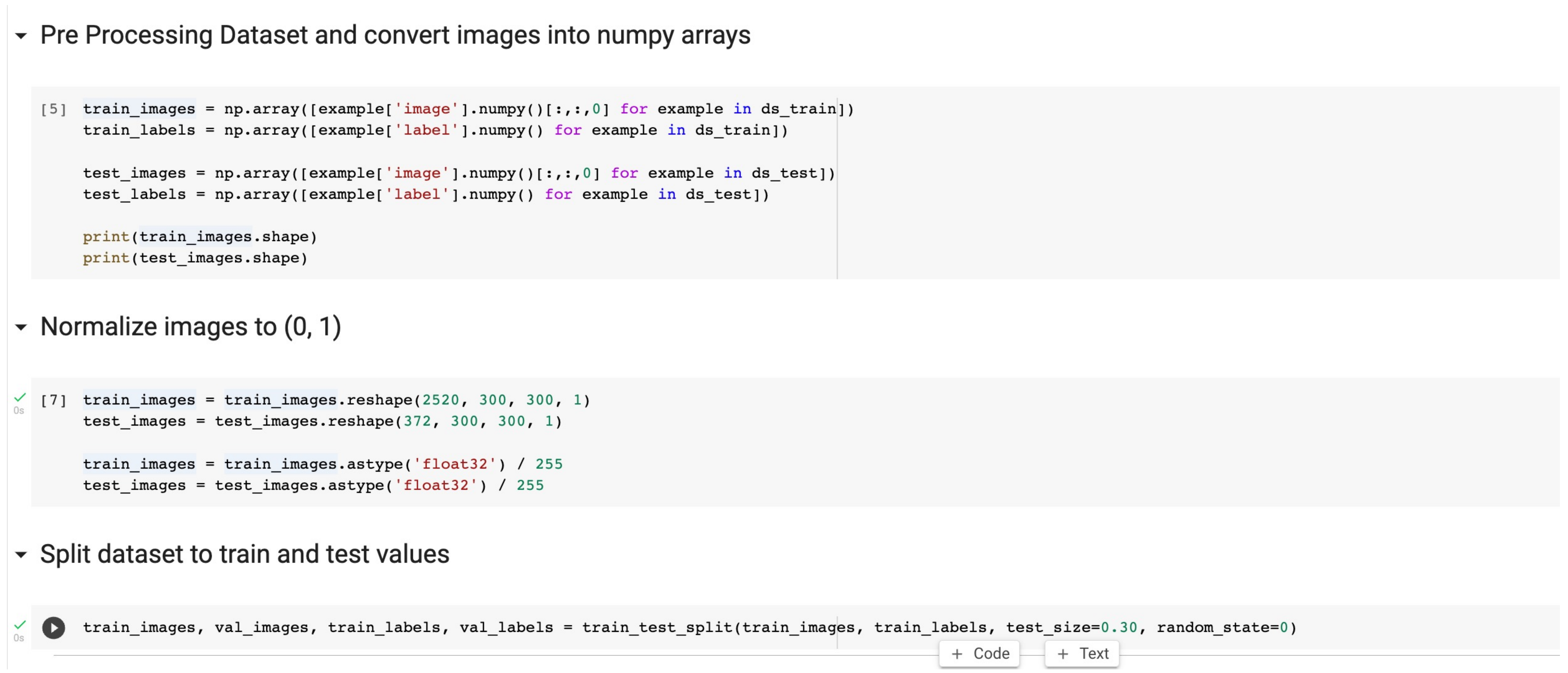
```
fig = tfds.show_examples(info, ds_train)
```

```
WARNING: For consistency with `tfds.load`, the `tfds.show_examples` signature has been modified from (info, ds) to (ds, info).
The old signature is deprecated and will be removed. Please change your call to `tfds.show_examples(ds, info)`
```



As mentioned in the question, we display some of the data and images in the dataset as examples.

Pre Processing



Pre Processing Dataset and convert images into numpy arrays

```
[5] train_images = np.array([example['image'].numpy()[:,:,0] for example in ds_train])
    train_labels = np.array([example['label'].numpy() for example in ds_train])

    test_images = np.array([example['image'].numpy()[:,:,0] for example in ds_test])
    test_labels = np.array([example['label'].numpy() for example in ds_test])

    print(train_images.shape)
    print(test_images.shape)
```

Normalize images to (0, 1)

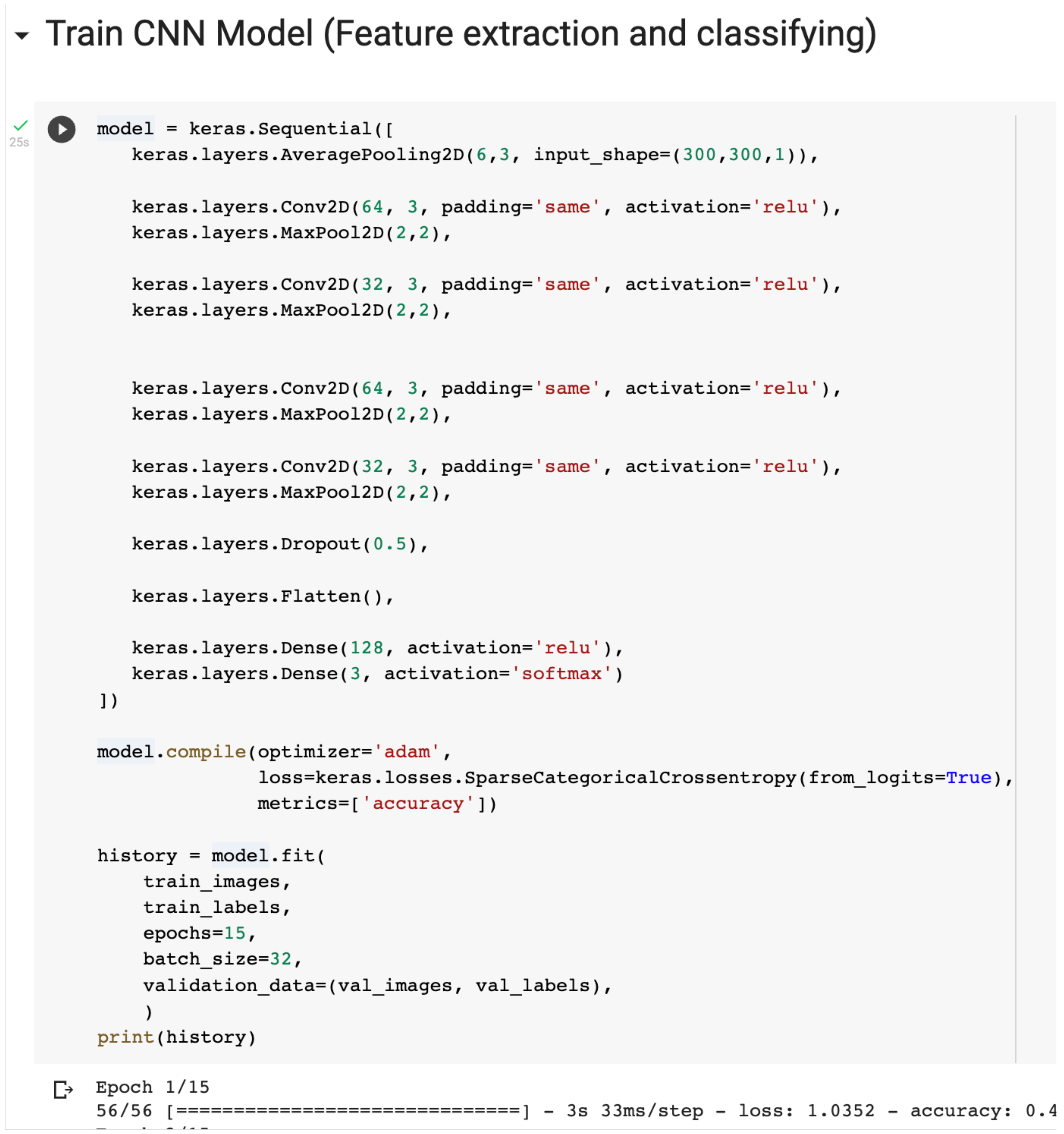
```
[7] train_images = train_images.reshape(2520, 300, 300, 1)
    test_images = test_images.reshape(372, 300, 300, 1)

    train_images = train_images.astype('float32') / 255
    test_images = test_images.astype('float32') / 255
```

Split dataset to train and test values

```
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.30, random_state=0)
```

In this stage, which is the processing stage before training the model, we store the data, which are our images, in nameless arrays so that processing can be done on them more easily.
In this step, the test and training data are separated along with their labels.
The point here is that we selected only the red color from the channel corresponding to the color of each image, and then attribute it to the single dimension that we added to the data matrix.
In the following, we will convert the type of brightness of the pixels from uint8 to float32 in order to apply the normalization operation.
We divide the brightness of each pixel by the maximum intensity, i.e. 255, so that they become between 0 and 1. We use the decimal type to prevent the division result from becoming zero.

We divide the training data dataset into two parts: training and validation data for a more accurate evaluation of the model.



Train CNN Model (Feature extraction and classifying)

```
model = keras.Sequential([
   keras.layers.AveragePooling2D(6,3, input_shape=(300,300,1)),

   keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),


   keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Dropout(0.5),

   keras.layers.Flatten(),

   keras.layers.Dense(128, activation='relu'),
   keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
    train_images,
    train_labels,
    epochs=15,
    batch_size=32,
    validation_data=(val_images, val_labels),
    )
print(history)
```

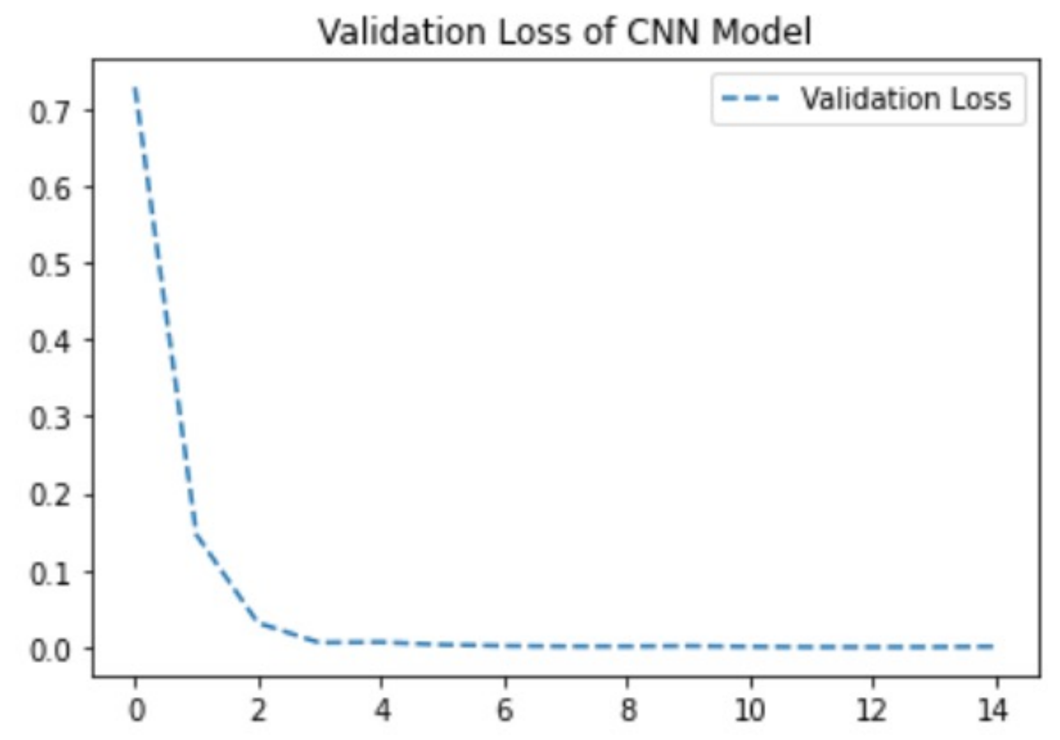
```
Epoch 1/15
56/56 [==============================] - 3s 33ms/step - loss: 1.0352 - accuracy: 0.4
```

Case 1:

Using Average Pooling, Conv, and Max Pool functions, we try to get a more accurate concept and abstract and a higher concept of input images in higher layers.
In this way, the number of dimensions decreases in higher layers and we get a higher concept of images and concepts of pixels.
We arrive from the concept of pixels to higher concepts such as edges and….
We use the Dropout function and half value to prevent Overfitting.
Also, for classification, we use the activation function of Soft Max, which has a possible meaning.

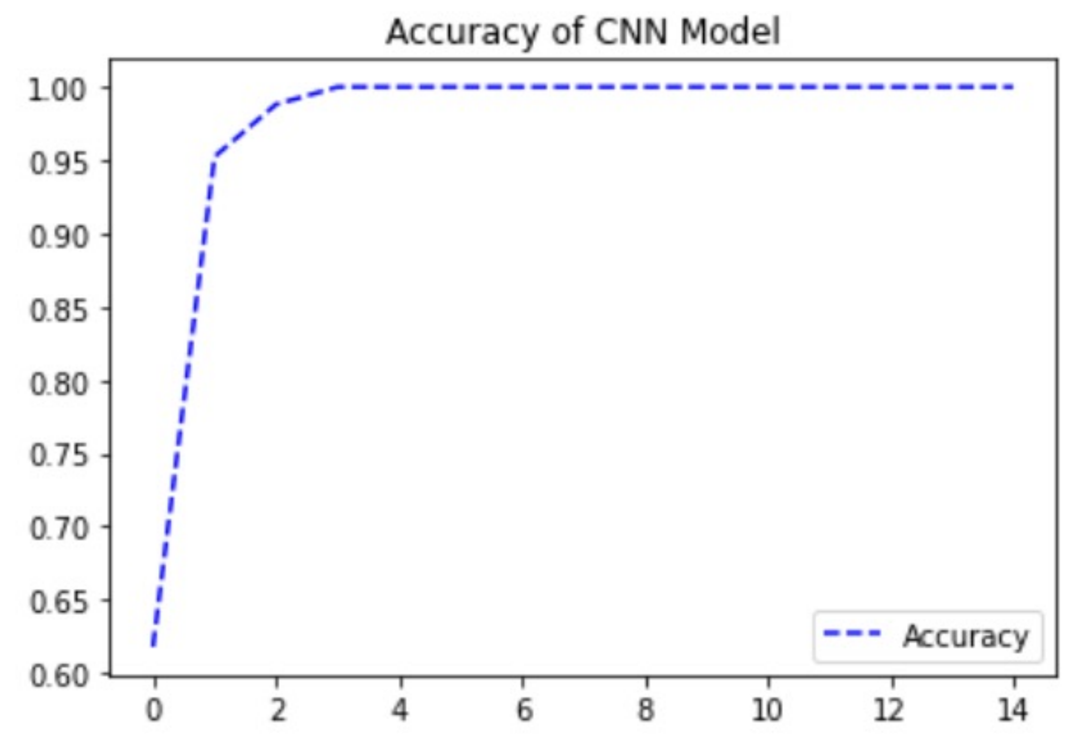
Validation Loss of CNN Model

```
[23] plt.title('Validation Loss of CNN Model')
     plt.plot(history.history['val_loss'], '--', label='Validation Loss')
     plt.legend()
     plt.show()
```

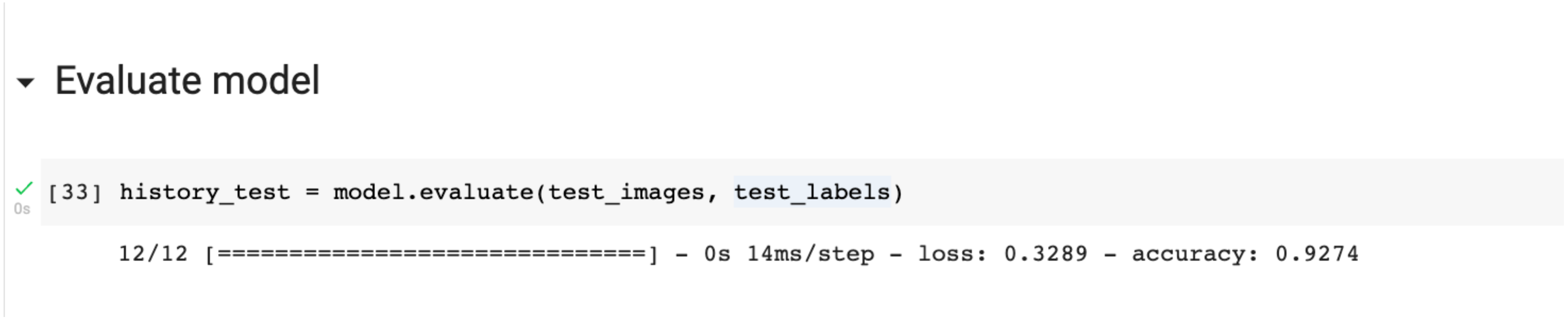


Validation Accuracy of CNN Model

```
plt.title('Accuracy of CNN Model')
plt.plot(history.history['val_accuracy'], '--', label='Accuracy', color='b')
plt.legend()
plt.show()
```



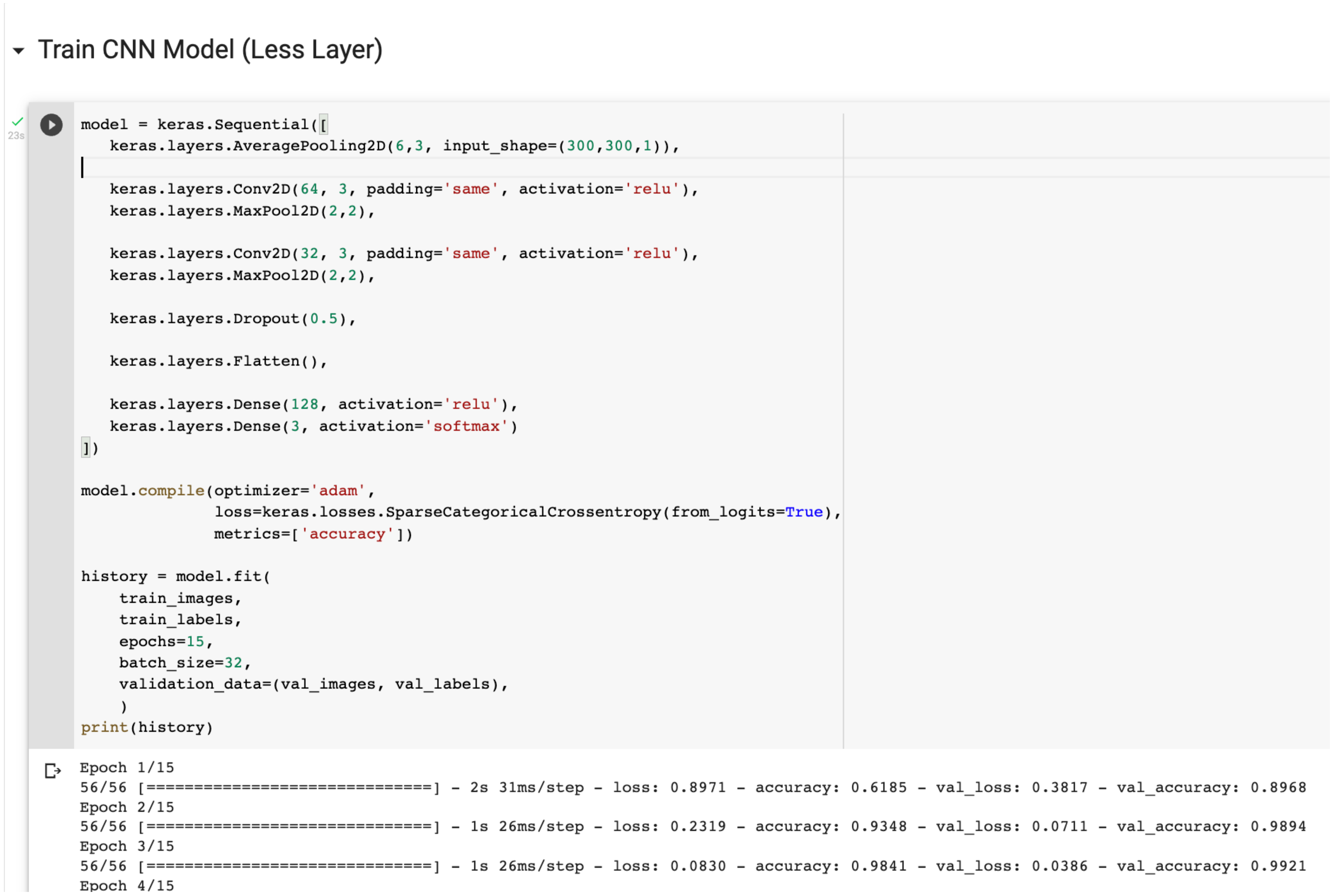
You can see the error and accuracy graphs of the trained model on the previous page.



Evaluate model

```
[33] history_test = model.evaluate(test_images, test_labels)

     12/12 [==============================] - 0s 14ms/step - loss: 0.3289 - accuracy: 0.9274
```

We check the accuracy of the model by testing the test data using the evaluate function and passing the test images and their labels.



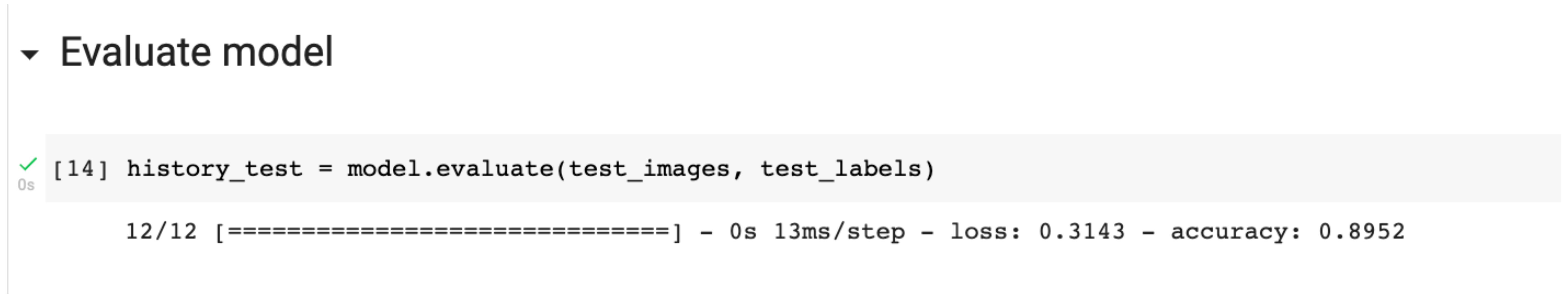
Train CNN Model (Less Layer)

```
model = keras.Sequential([
    keras.layers.AveragePooling2D(6,3, input_shape=(300,300,1)),

    keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    keras.layers.MaxPool2D(2,2),

    keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    keras.layers.MaxPool2D(2,2),

    keras.layers.Dropout(0.5),

    keras.layers.Flatten(),

    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
    train_images,
    train_labels,
    epochs=15,
    batch_size=32,
    validation_data=(val_images, val_labels),
    )
print(history)
```

```
Epoch 1/15
56/56 [==============================] - 2s 31ms/step - loss: 0.8971 - accuracy: 0.6185 - val_loss: 0.3817 - val_accuracy: 0.8968
Epoch 2/15
56/56 [==============================] - 1s 26ms/step - loss: 0.2319 - accuracy: 0.9348 - val_loss: 0.0711 - val_accuracy: 0.9894
Epoch 3/15
56/56 [==============================] - 1s 26ms/step - loss: 0.0830 - accuracy: 0.9841 - val_loss: 0.0386 - val_accuracy: 0.9921
Epoch 4/15
```

Case 2:

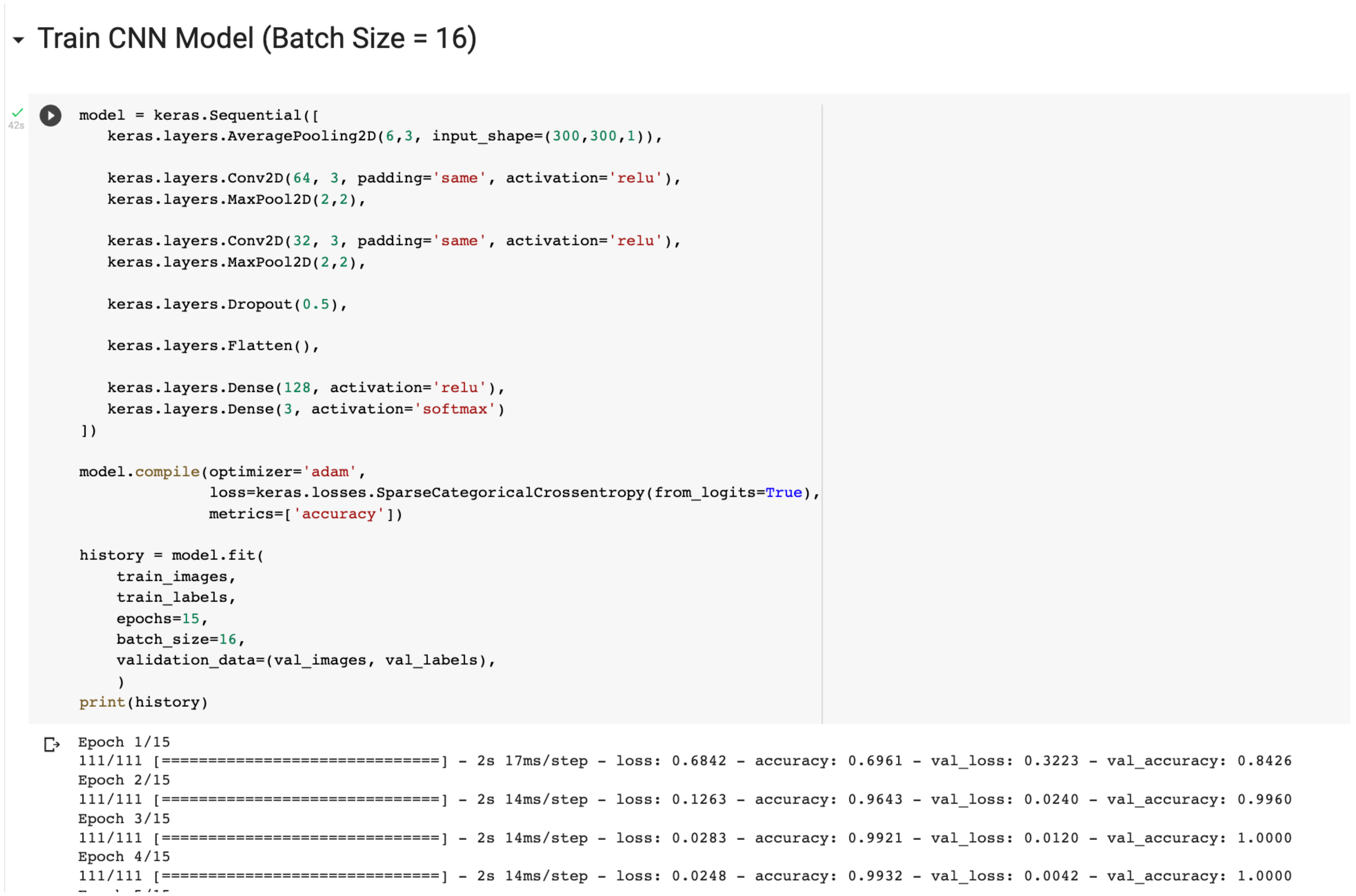
To check the same model with less number of layers, we train another model this time with 2 convolutional layers.
We consider the rest of the parameters constant.



Evaluate model

```
[14] history_test = model.evaluate(test_images, test_labels)

     12/12 [==============================] - 0s 13ms/step - loss: 0.3143 - accuracy: 0.8952
```

We check the accuracy of the model by testing the test data using the evaluate function and passing the test images and their labels.
You can see that the accuracy of our model has decreased by a very small amount compared to the previous model.
Of course, it should be taken into account that these accuracies may increase and decrease with each training of the model.



Train CNN Model (Batch Size = 16)

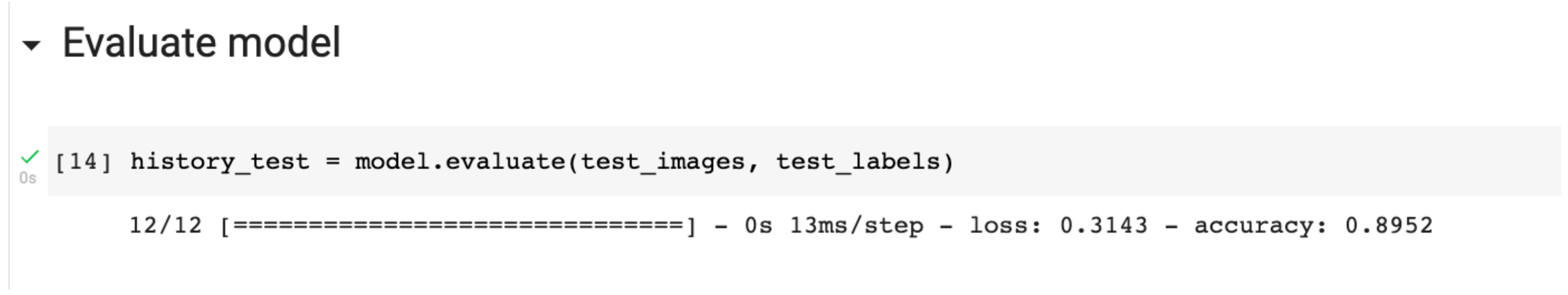
```
model = keras.Sequential([
    keras.layers.AveragePooling2D(6,3, input_shape=(300,300,1)),

    keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    keras.layers.MaxPool2D(2,2),

    keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    keras.layers.MaxPool2D(2,2),

    keras.layers.Dropout(0.5),

    keras.layers.Flatten(),

    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
     train_images,
     train_labels,
     epochs=15,
     batch_size=16,
     validation_data=(val_images, val_labels),
     )
print(history)
```

```
Epoch 1/15
111/111 [==============================] - 2s 17ms/step - loss: 0.6842 - accuracy: 0.6961 - val_loss: 0.3223 - val_accuracy: 0.8426
Epoch 2/15
111/111 [==============================] - 2s 14ms/step - loss: 0.1263 - accuracy: 0.9643 - val_loss: 0.0240 - val_accuracy: 0.9960
Epoch 3/15
111/111 [==============================] - 2s 14ms/step - loss: 0.0283 - accuracy: 0.9921 - val_loss: 0.0120 - val_accuracy: 1.0000
Epoch 4/15
111/111 [==============================] - 2s 14ms/step - loss: 0.0248 - accuracy: 0.9932 - val_loss: 0.0042 - val_accuracy: 1.0000
```

Case 3:

To check the same model with the number of samples of each package, we train the model again, this time with the number of 16 packages.

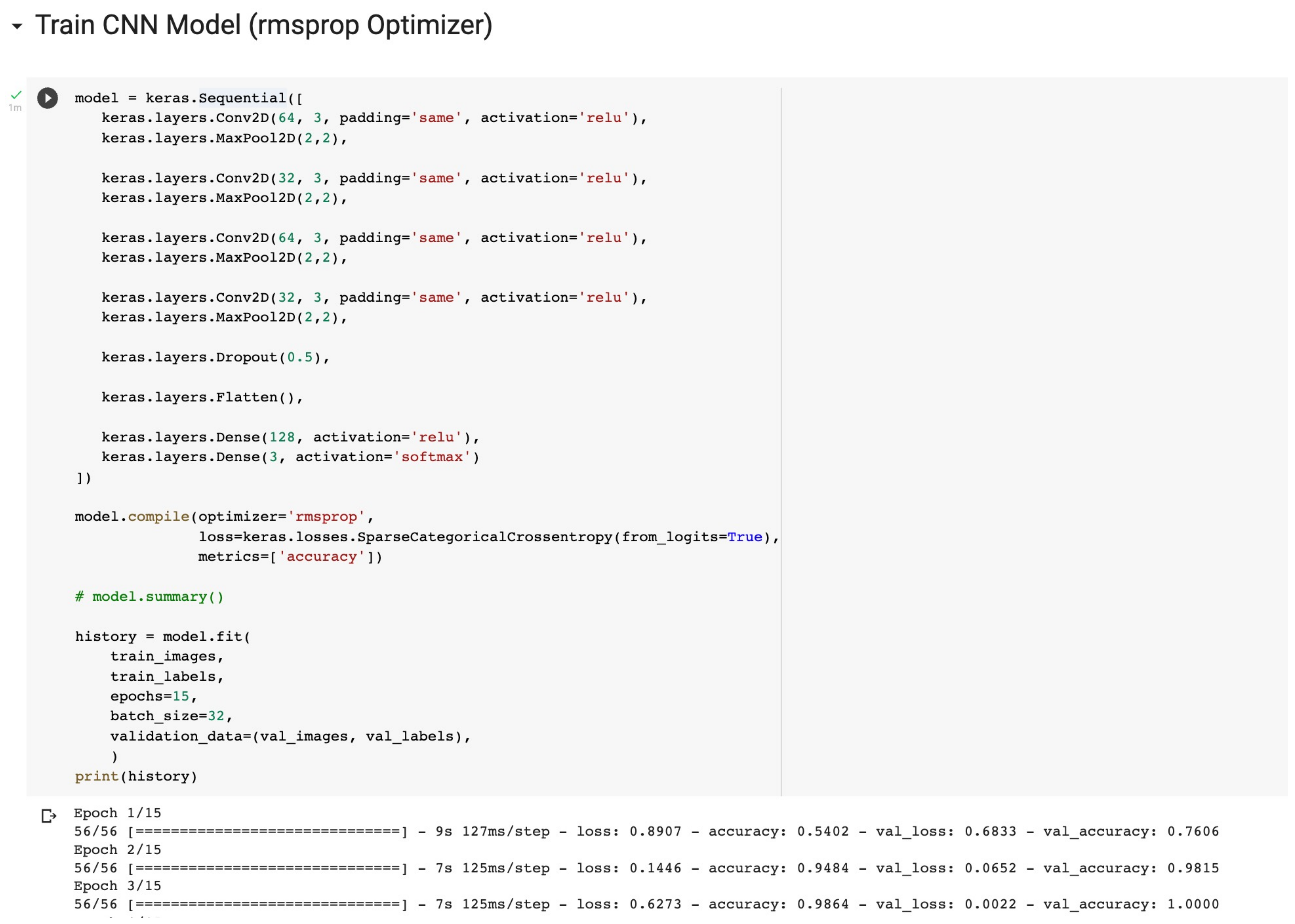
We kept the rest of the parameters constant.



```
history_test = model.evaluate(test_images, test_labels)
```

```
12/12 [==============================] - 0s 13ms/step - loss: 0.3143 - accuracy: 0.8952
```

We check the accuracy of the model by testing the test data using the evaluate function and passing the test images and their labels.
You can see that the accuracy of our model has decreased by a very small amount compared to the previous model.
Of course, it should be taken into account that these accuracies may increase and decrease with each training of the model.



Train CNN Model (rmsprop Optimizer)

```
model = keras.Sequential([
   keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
   keras.layers.MaxPool2D(2,2),

   keras.layers.Dropout(0.5),

   keras.layers.Flatten(),

   keras.layers.Dense(128, activation='relu'),
   keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='rmsprop',
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# model.summary()

history = model.fit(
    train_images,
    train_labels,
    epochs=15,
    batch_size=32,
    validation_data=(val_images, val_labels),
    )
print(history)
```

```
Epoch 1/15
56/56 [==============================] - 9s 127ms/step - loss: 0.8907 - accuracy: 0.5402 - val_loss: 0.6833 - val_accuracy: 0.7606
Epoch 2/15
56/56 [==============================] - 7s 125ms/step - loss: 0.1446 - accuracy: 0.9484 - val_loss: 0.0652 - val_accuracy: 0.9815
Epoch 3/15
56/56 [==============================] - 7s 125ms/step - loss: 0.6273 - accuracy: 0.9864 - val_loss: 0.0022 - val_accuracy: 1.0000
```

Case 4:

To check the same model with Rmsprop optimization, instead of Adam, we train another model with Rmsprop optimization.

We consider the rest of the parameters constant.

```
12/12 [==============================] - 0s 12ms/step - loss: 2.4549 -
accuracy: 0.7419
```

We check the accuracy of the model by testing the test data using the evaluate function and passing the test images and their labels.
You can see that the accuracy of our model has decreased somewhat compared to the previous model. And it turns out that Adam's optimizer has shown better performance.
Of course, it should be taken into account that these accuracies may increase and decrease with each training of the model.