NEURAL NETWORK & DEEP LEARNING

ASSIGNMENT 8

Name : SRAVYA REDDY PILLI

Student ID : 700747154

Git hub Link: https://github.com/09sravyareddy/NNDL-ICP8

Video link:
https://drive.google.com/file/d/1cZIKBmOUYRK3HT3J6wMcEwh8A5RsQc6v/view?usp=drive_link

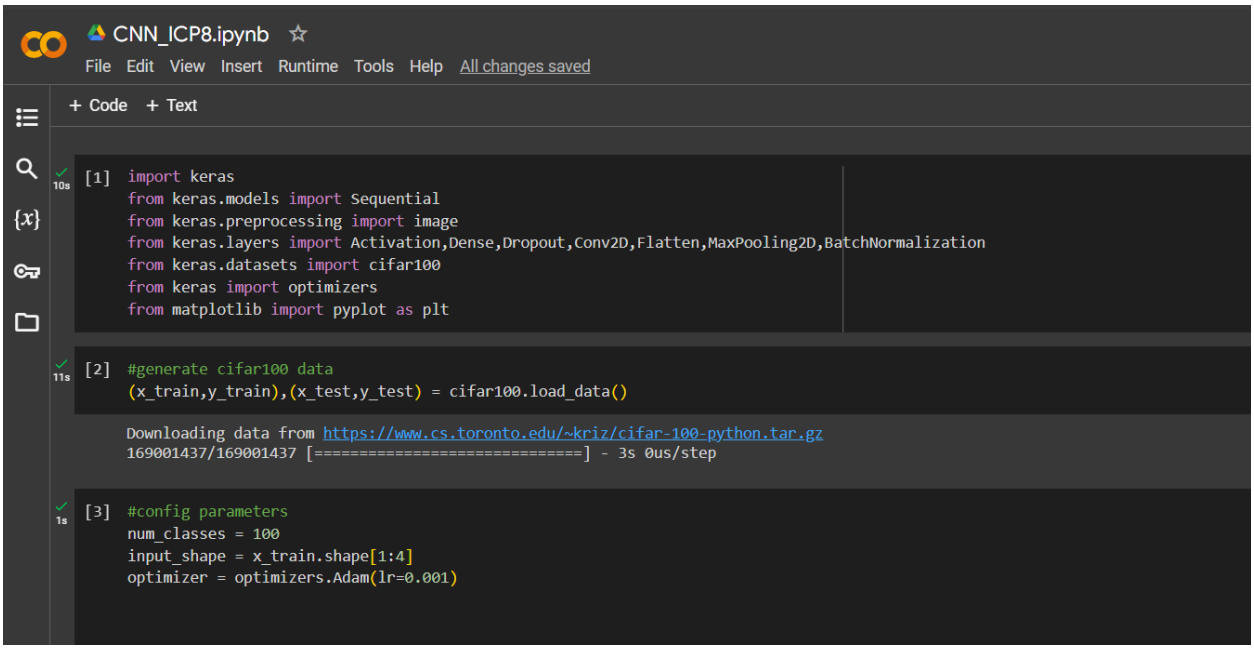
Lesson Overview:
In this lesson, we are going to discuss Image classification with CNN.

Use Case Description:
LeNet5, AlexNet, Vgg16, Vgg19
1. Training the model
2. Evaluating the model

Programming elements:
1. About CNN
2. Hyperparameters of CNN
3. Image classification with CNN

In class programming:

1. Tune hyperparameter and make necessary addition to the baseline model to improve validation accuracy and reduce validation loss.
2. Provide logical description of which steps lead to improved response and what was its impact on architecture behavior.
3. Create at least two more visualizations using matplotlib (Other than provided in the source file)
4. Use dataset of your own choice and implement baseline models provided.
5. Apply modified architecture to your own selected dataset and train it.
6. Evaluate your model on testing set.
7. Save the improved model and use it for prediction on testing data
8. Provide plot of confusion matric
9. Provide Training and testing Loss and accuracy plots in one plot using subplot command and history object.
10. Provide at least two more visualizations reflecting your solution.
11. Provide logical description of which steps lead to improved response for new dataset when compared with baseline model and enhance architecture and what was its impact on architecture behavior.

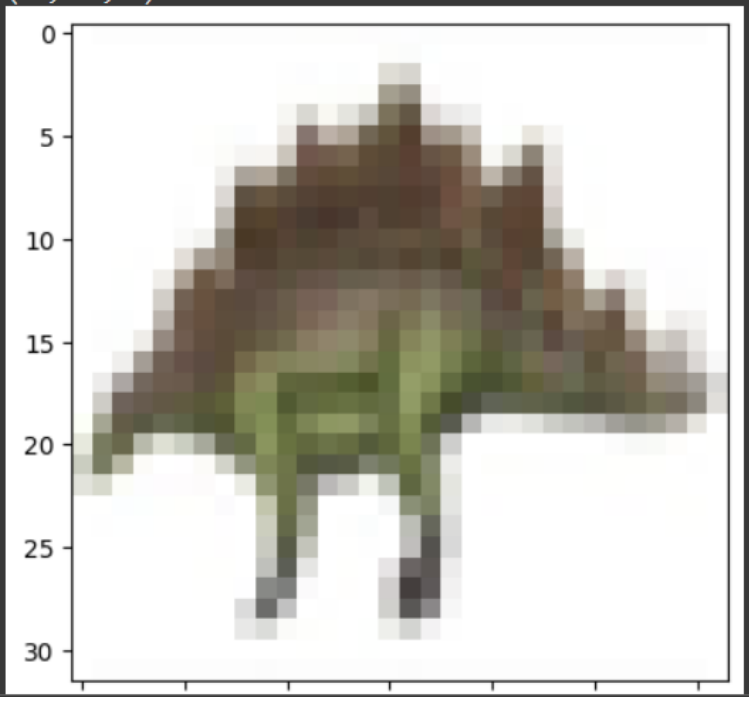

CNN_ICP8.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[1] import keras
from keras.models import Sequential
from keras.preprocessing import image
from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D,BatchNormalization
from keras.datasets import cifar100
from keras import optimizers
from matplotlib import pyplot as plt
[2] #generate cifar100 data
(x_train,y_train),(x_test,y_test) = cifar100.load_data()
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169001437/169001437 [==============================] - 3s 0us/step
[3] #config parameters
num_classes = 100
input_shape = x_train.shape[1:4]
optimizer = optimizers.Adam(lr=0.001)

+ Code + Text

```
[4] one_hot_y_train = keras.utils.to_categorical(y_train, num_classes=num_classes)
    one_hot_y_test = keras.utils.to_categorical(y_test, num_classes=num_classes)
```

```
# check data
plt.imshow(x_train[1])
print(x_train[1].shape)
```

```
(32, 32, 3)
```



+ Code + Text

[6] 0s

```
# build model(similar to VGG16, only change the input and output shape)
model = Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
```

+ Code + Text

0s

```
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(2048, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

+ Code + Text

[7] `model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])`

`model.summary()`

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        1792

 batch_normalization (Batch  (None, 32, 32, 64)        256
 Normalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        36928

 batch_normalization_1 (Bat  (None, 32, 32, 64)        256
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 64)        0
 D)

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       73856

 batch_normalization_2 (Bat  (None, 16, 16, 128)       512
 chNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584

 batch_normalization_3 (Bat  (None, 16, 16, 128)       512
```

+ Code + Text

```
 dense_1 (Dense)             (None, 2048)              8390656

 dense_2 (Dense)             (None, 1024)              2098176

 dropout_5 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 100)               102500

 activation (Activation)     (None, 100)               0

=================================================================
Total params: 27424164 (104.61 MB)
Trainable params: 27415716 (104.58 MB)
Non-trainable params: 8448 (33.00 KB)
_________________________________________________________________
```

`history = model.fit(x=x_train, y=one_hot_y_train, batch_size=128, epochs=30, validation_split=0.1)`

```
Epoch 1/30
352/352 [==============================] - 47s 88ms/step - loss: 4.2764 - accuracy: 0.0397 - val_loss: 4.7306 - val_accuracy: 0.0348
Epoch 2/30
352/352 [==============================] - 28s 81ms/step - loss: 3.9276 - accuracy: 0.0656 - val_loss: 4.8051 - val_accuracy: 0.0476
Epoch 3/30
352/352 [==============================] - 30s 84ms/step - loss: 3.7085 - accuracy: 0.0925 - val_loss: 3.9138 - val_accuracy: 0.1050
Epoch 4/30
352/352 [==============================] - 29s 83ms/step - loss: 3.4520 - accuracy: 0.1316 - val_loss: 3.3779 - val_accuracy: 0.1558
Epoch 5/30
352/352 [==============================] - 29s 82ms/step - loss: 3.2011 - accuracy: 0.1785 - val_loss: 3.2389 - val_accuracy: 0.1918
Epoch 6/30
352/352 [==============================] - 29s 82ms/step - loss: 2.9446 - accuracy: 0.2274 - val_loss: 2.9098 - val_accuracy: 0.2486
Epoch 7/30
352/352 [==============================] - 29s 83ms/step - loss: 2.7457 - accuracy: 0.2662 - val_loss: 2.7727 - val_accuracy: 0.2702
```

✓ 4s completed at 11:32 PM

+ Code + Text

```
Epoch 29/30
352/352 [==============================] - 28s 81ms/step - loss: 0.7115 - accuracy: 0.7894 - val_loss: 2.3604 - val_accuracy: 0.5046
Epoch 30/30
352/352 [==============================] - 28s 81ms/step - loss: 0.6774 - accuracy: 0.7990 - val_loss: 2.4038 - val_accuracy: 0.5052
```

[10]

```
# evaluate
print(model.metrics_names)
model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
['loss', 'accuracy']
20/20 [==============================] - 7s 176ms/step - loss: 2.2925 - accuracy: 0.5229
[2.292543888092041, 0.5228999853134155]
```

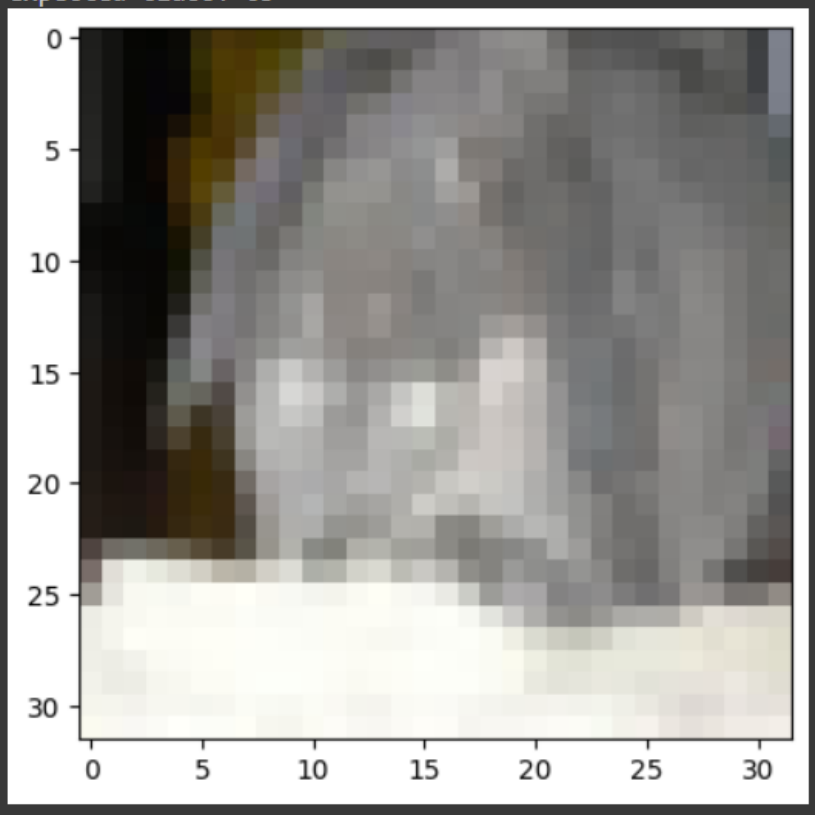
```
model.save("keras-VGG16-cifar10.h5")
plt.imshow(x_test[1000])

result = model.predict(x_test[1000:1001]).tolist()
predict = 0
expect = y_test[1000][0]
for i,_ in enumerate(result[0]):
   if result[0][i] > result[0][predict]:
     predict = i
print("predict class:",predict)
print("expected class:",expect)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file
  saving_api.save_model(
```

+ Code + Text

print("expected class:",expect)

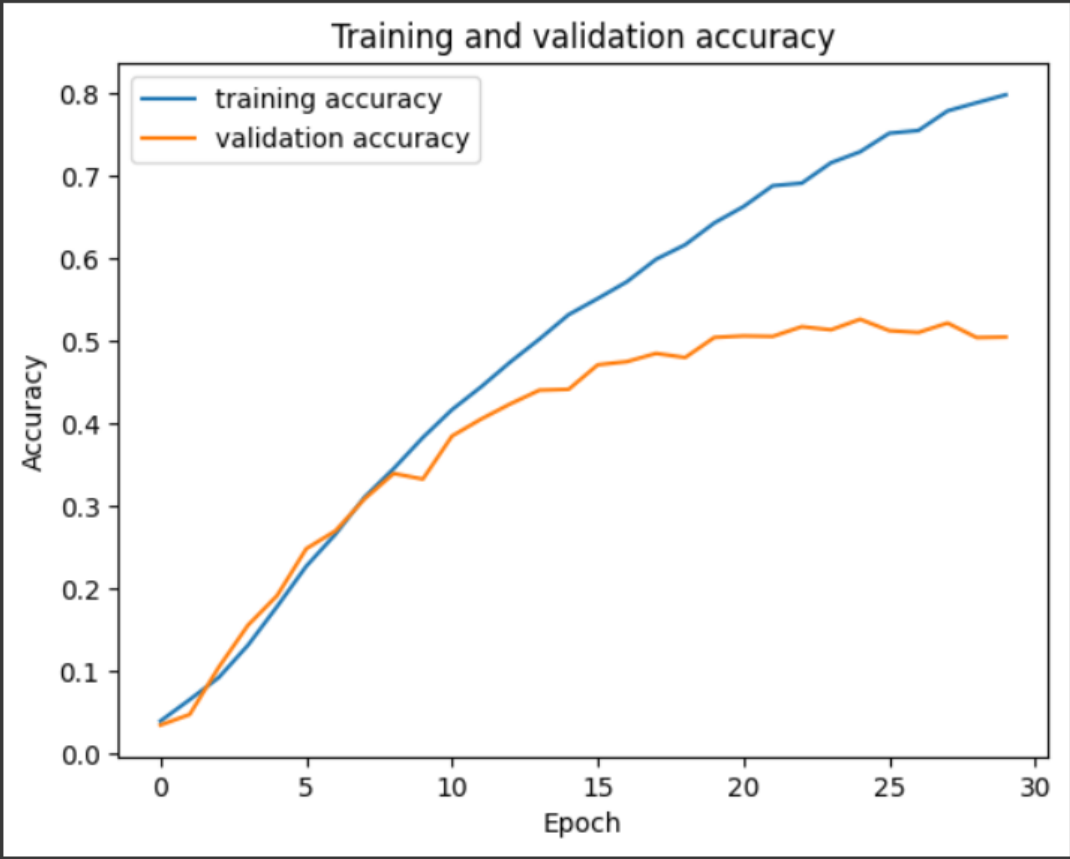
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: U
  saving_api.save_model(
1/1 [==============================] - 1s 1s/step
predict class: 63
expected class: 65
```



```
[12] # save model
1s   model.save("keras-VGG16-cifar10.h5")
```

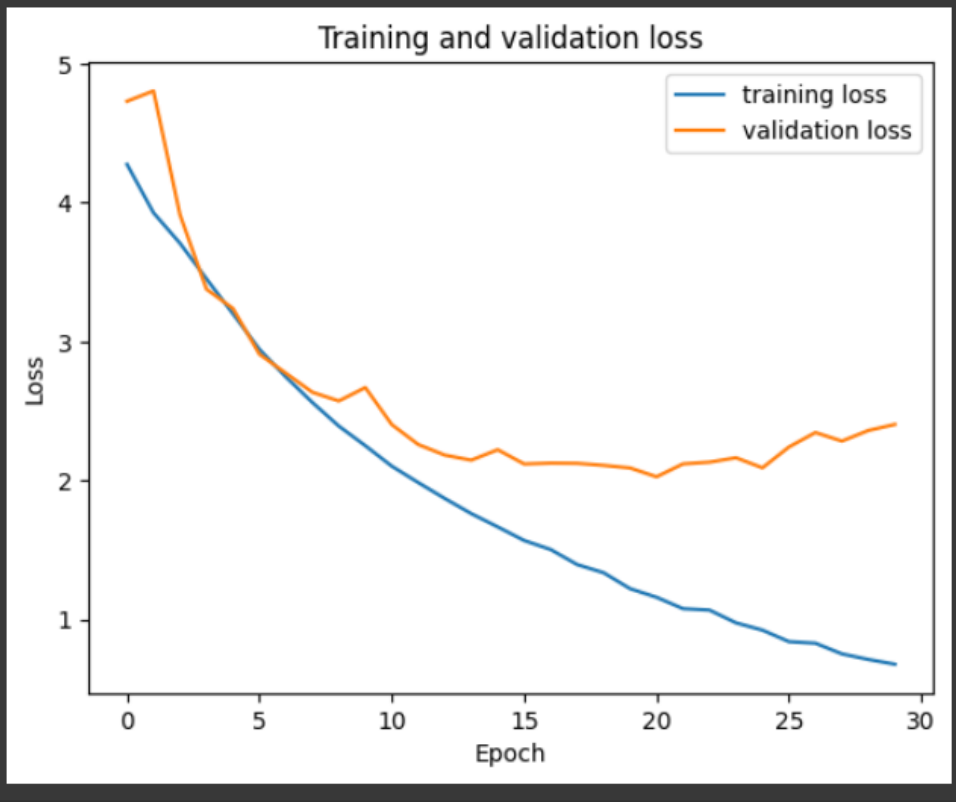
```
0s
#plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

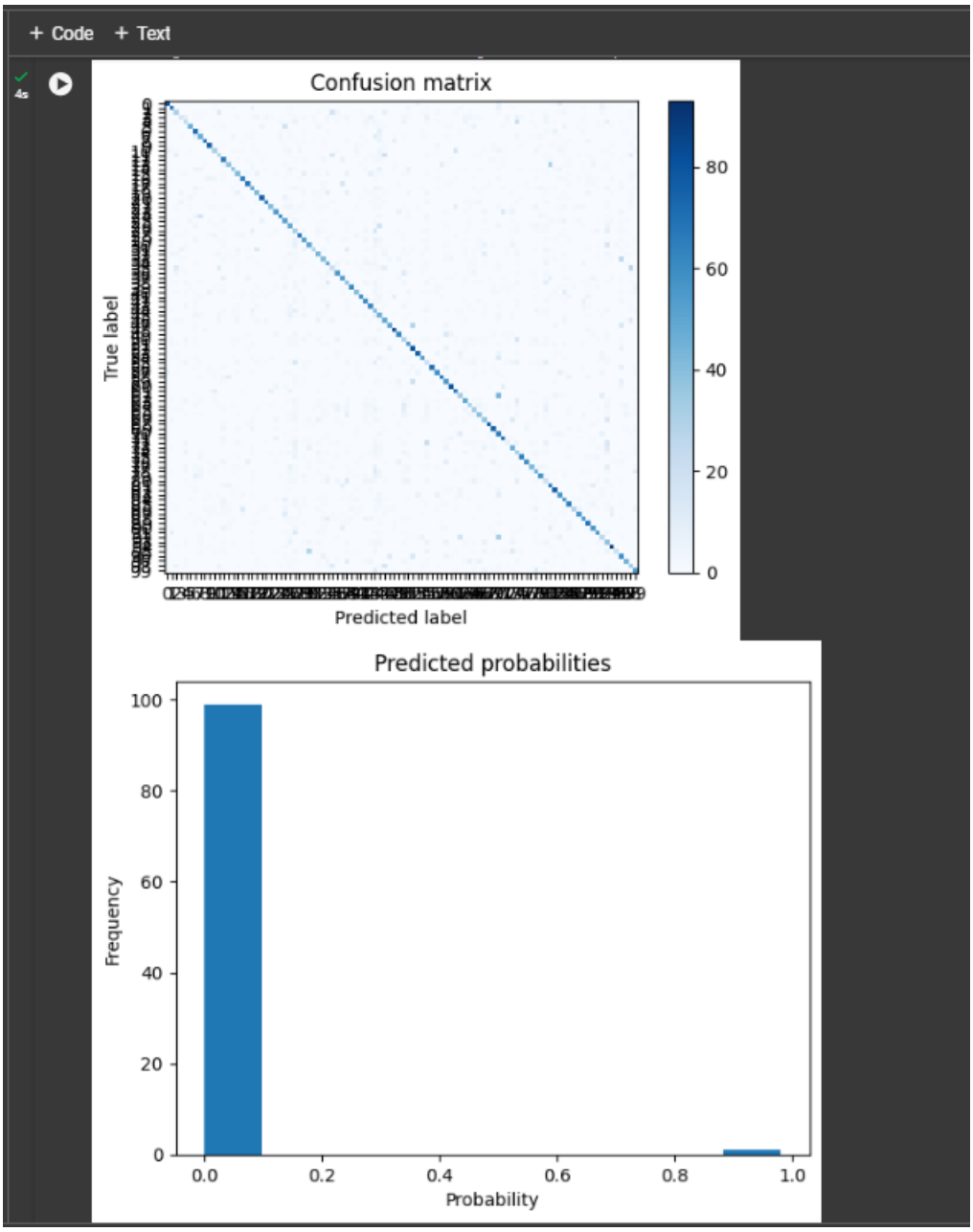


+ Code + Text

```
#plot the training and validation loss
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```python
import numpy as np
from sklearn.metrics import confusion_matrix

# calculate the confusion matrix
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = y_test.ravel()
cm = confusion_matrix(y_true, y_pred_classes)

# plot the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, range(num_classes))
plt.yticks(tick_marks, range(num_classes))
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# plot a histogram of the predicted probabilities for a sample image
plt.hist(y_pred[1000])
plt.title('Predicted probabilities')
plt.xlabel('Probability')
plt.ylabel('Frequency')
plt.show()
```


+ Code
+ Text
4s
Confusion matrix
True label
Predicted label
80
60
40
20
0
Predicted probabilities
Frequency
Probability
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0