Exercises4

November 20, 2020

1 Exercise set 4

1.1 Answered to all questions (1-5).

2 Question 1

Logistic loss function:

$$l(\mathbf{w}) = \sum_{n=0}^{N-1} ln(1 + exp(-y_n \mathbf{w^T x_n}))$$

Gradient of $l(\mathbf{w})$:

$$\begin{aligned} l'(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} \sum_{n=0}^{N-1} ln(1 + exp(-y_n \mathbf{w^T x_n})) \\ &= \sum_{n=0}^{N-1} \frac{\partial}{\partial \mathbf{w}} ln(1 + exp(-y_n \mathbf{w^T x_n})) \end{aligned} \tag{1}$$

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} ln(1 + exp(-y_n \mathbf{w^T x_n})) &= \frac{1}{1 + exp(-y_n \mathbf{w^T x_n})} * \frac{\partial}{\partial \mathbf{w}} 1 + exp(-y_n \mathbf{w^T x_n}) \\ &= \frac{1}{1 + exp(-y_n \mathbf{w^T x_n})} * \frac{\partial}{\partial \mathbf{w}} exp(-y_n \mathbf{w^T x_n}) \end{aligned} \tag{2}$$

$$\frac{\partial}{\partial \mathbf{w}} exp(-y_n \mathbf{w^T x_n}) = exp(-y_n \mathbf{w^T x_n}) * \frac{\partial}{\partial \mathbf{w}} - y_n \mathbf{w^T x_n} \tag{3}$$

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} - y_n \mathbf{w^T x_n} &= -y_n \frac{\partial}{\partial \mathbf{w}} \mathbf{w^T x_n} \\ &= -y_n \mathbf{x_n} \end{aligned} \tag{4}$$

Therefore:

$$(1), (2), (3), (4) \Rightarrow l'(\mathbf{w}) = \sum_{n=0}^{N-1} \frac{exp(-y_n \mathbf{w^T x_n})}{1 + exp(-y_n \mathbf{w^T x_n})} (-y_n \mathbf{x_n})$$

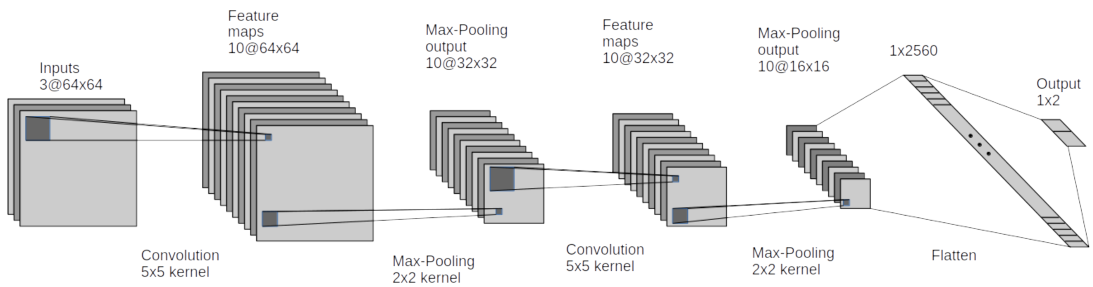
Gradient of $C * \mathbf{w^T w}$:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} C * \mathbf{w^T w} &= C * \frac{\partial}{\partial \mathbf{w}} \mathbf{w^T w} \\ &= C * \frac{\partial}{\partial \mathbf{w}} \sum w_i^2 \\ &= C * (2w_0, \cdots, 2w_P)^T \\ &= 2C\mathbf{w}\end{aligned}$$

Therefore, the gradient of the *L2-regularized logistic loss* is:

$$l'(\mathbf{w}) = \sum_{n=0}^{N-1} \frac{exp(-y_n \mathbf{w^T x_n})}{1 + exp(-y_n \mathbf{w^T x_n})} (-y_n \mathbf{x_n}) + 2C\mathbf{w}$$

3 Question 2



a)

b) Number of parameters in each layer

Input layer = 0

No learnable parameters.

First convolution $= 5 * 5 * 3 * 10 + 10 = 760$

Convolution window (weight matrix) of size 5x5. Therefore 5*5 learnable parameters.

Need to learn a unique conv. window for each feature map in the input. Therefore *3.

Need to learn a unique conv. window parameters for each feature map in the output. Therefore *

Bias for each feature map in output. Therefore +10.

Max-Pool = 0

No learnable parameters. Just shrinks the size of each feature map.

Second convolution $= 5 * 5 * 10 * 10 + 10 = 2510$

Convolution window (weight matrix) of size 5x5. Therefore 5*5 learnable parameters.

Need to learn a unique conv. window for each feature map in the input. Therefore *10.

Need to learn a unique conv. window parameters for each feature map in the output. Therefore *

```
Bias for each feature map in output. Therefore +10.
```

Max-Pool = 0

Flatten = 0

```
No learnable parameters. Just reduces the dimensionality of the input.
```

Output layer $= 2560 * 2 + 2 = 5122$

```
2560 neurons in previous layer connected to output layer of size of 2. Therefore 2560*2 weight
```

```
Also, the neurons in the output layer have bias term each. Therefore +2 parameters.
```

Total number of parameters in network = 8392

4 Question 3

[1]:
```python
import numpy as np
import os
# Load the data
os.chdir('/home/tuomas/Python/DATA.ML.200/Ex4')
dataX = np.loadtxt('X.csv', delimiter=',')
dataY = np.loadtxt('y.csv', delimiter=',')
```

[30]:
```python
# Gradient descent

def add_bias(w, X, b):
    new_w = w.tolist()
    new_w.append(b)
    new_X = np.ones((X.shape[0], X.shape[1]+1))
    new_X[:,:-1] = X
    return np.array(new_w), new_X

def log_loss(w):
    loss = 0
    for n in range(X.shape[0]):
        x = X[n]
        y = dataY[n]
        loss += np.log(1 + np.exp(-y * w@x))

    return loss

def loss_gradient(w):
    grad = np.zeros(w.shape)
    for n in range(X.shape[0]):
        x = X[n]
        y = dataY[n]
        num = np.exp(-y * w@x)
```

```
        denom = 1 + np.exp(-y * w@x)
        grad += (num/denom)*(-y*x)

    return grad

def predict(w):
    preds = []
    for n in range(X.shape[0]):
        x = X[n]
        class_1 = (1 / (1 + np.exp(-w@x))) > 0.5
        if class_1:
            preds.append(1)
        else:
            preds.append(-1)

    return np.array(preds)

# Initialize w as random
w = np.random.randn(2)
# Add bias term to the model (init. as random as well)
w, X = add_bias(w, dataX, np.random.randn(1)[0])

# Set learning rate
e = 0.01
iterations = 100
old_ws = []
old_accs = []
for i in range(iterations):
    # Update w
    w = w - e*loss_gradient(w)
    # Print info
    print('Iteration {}. w = {}, log-loss = {}'.format(i+1, w[:2], log_loss(w)))
    # Calculate accuracy
    y_hat = predict(w)
    acc = np.sum(y_hat == dataY) / 400.0
    old_accs.append(acc)
    old_ws.append(w)
```

```
Iteration 1. w = [-1.40993347  1.51911066], log-loss = 120.0308246119129
Iteration 2. w = [-0.97233901  1.62664125], log-loss = 100.6582735812171
Iteration 3. w = [-0.64919303  1.6856105 ], log-loss = 91.0521173245905
Iteration 4. w = [-0.40953901  1.72960377], log-loss = 85.81339724422338
Iteration 5. w = [-0.2272205   1.76830622], log-loss = 82.54460203484516
Iteration 6. w = [-0.08380088  1.80327307], log-loss = 80.25616945229795
Iteration 7. w = [0.03270673 1.83460917], log-loss = 78.53482482724279
Iteration 8. w = [0.12982235 1.86261584], log-loss = 77.19063349610738
Iteration 9. w = [0.21229145 1.88783656], log-loss = 76.12211418603785
```

```
Iteration 10. w = [0.28322348 1.91085883], log-loss = 75.26596446102715
Iteration 11. w = [0.34477385 1.93218611], log-loss = 74.57762982649841
Iteration 12. w = [0.39852951 1.95219344], log-loss = 74.02328617845696
Iteration 13. w = [0.44572056 1.97113183], log-loss = 73.57624251642194
Iteration 14. w = [0.48733564 1.98915342], log-loss = 73.21511706889832
Iteration 15. w = [0.52418667 2.00634089], log-loss = 72.92273788286835
Iteration 16. w = [0.55694791 2.02273332], log-loss = 72.6853525537232
Iteration 17. w = [0.58618244 2.03834596], log-loss = 72.4919922323916
Iteration 18. w = [0.61236196 2.05318369], log-loss = 72.33393785996226
Iteration 19. w = [0.63588265 2.0672494 ], log-loss = 72.20427191194483
Iteration 20. w = [0.65707838 2.08054867], log-loss = 72.09750761745924
Iteration 21. w = [0.67623158 2.0930918 ], log-loss = 72.00928774019741
Iteration 22. w = [0.69358228 2.10489443], log-loss = 71.93614369723193
Iteration 23. w = [0.70933553 2.11597715], log-loss = 71.87530522180367
Iteration 24. w = [0.72366743 2.12636473], log-loss = 71.82455117990301
Iteration 25. w = [0.73673002 2.13608519], log-loss = 71.78209319253463
Iteration 26. w = [0.74865529 2.14516879], log-loss = 71.74648502385945
Iteration 27. w = [0.75955841 2.15364726], log-loss = 71.71655201145197
Iteration 28. w = [0.76954031 2.16155299], log-loss = 71.69133599956866
Iteration 29. w = [0.77868989 2.16891844], log-loss = 71.67005223514175
Iteration 30. w = [0.78708574 2.17577568], log-loss = 71.65205549424364
Iteration 31. w = [0.79479763 2.18215603], log-loss = 71.6368133431265
Iteration 32. w = [0.80188769 2.18808975], log-loss = 71.6238849304589
Iteration 33. w = [0.80841144 2.19360589], log-loss = 71.61290408447032
Iteration 34. w = [0.81441865 2.19873217], log-loss = 71.60356577563259
Iteration 35. w = [0.81995402 2.20349487], log-loss = 71.59561522321077
Iteration 36. w = [0.82505784 2.20791885], log-loss = 71.58883908914257
Iteration 37. w = [0.82976648 2.2120275 ], log-loss = 71.58305832814172
Iteration 38. w = [0.83411284 2.21584281], log-loss = 71.57812235847256
Iteration 39. w = [0.83812675 2.21938536], log-loss = 71.57390429089816
Iteration 40. w = [0.84183532 2.22267441], log-loss = 71.57029700941074
Iteration 41. w = [0.84526319 2.22572794], log-loss = 71.56720994064261
Iteration 42. w = [0.8484328  2.22856271], log-loss = 71.56456638242473
Iteration 43. w = [0.85136466 2.23119436], log-loss = 71.56230128812578
Iteration 44. w = [0.85407746 2.2336374 ], log-loss = 71.56035942390605
Iteration 45. w = [0.85658834 2.23590536], log-loss = 71.55869383215281
Iteration 46. w = [0.85891296 2.23801081], log-loss = 71.55726454714234
Iteration 47. w = [0.86106568 2.23996541], log-loss = 71.55603751911839
Iteration 48. w = [0.8630597  2.24178001], log-loss = 71.55498371108892
Iteration 49. w = [0.86490711 2.24346468], log-loss = 71.55407833914339
Iteration 50. w = [0.86661905 2.24502875], log-loss = 71.55330023233239
Iteration 51. w = [0.86820573 2.2464809 ], log-loss = 71.55263129238745
Iteration 52. w = [0.86967658 2.24782918], log-loss = 71.55205603699386
Iteration 53. w = [0.87104028 2.24908105], log-loss = 71.55156121313833
Iteration 54. w = [0.87230482 2.25024343], log-loss = 71.55113546933262
Iteration 55. w = [0.87347756 2.25132277], log-loss = 71.5507690774014
Iteration 56. w = [0.87456532 2.25232501], log-loss = 71.550453696061
Iteration 57. w = [0.87557437 2.25325569], log-loss = 71.55018216979416
```

```
Iteration 58. w = [0.87651051 2.25411995], log-loss = 71.54994835757547
Iteration 59. w = [0.87737909 2.25492254], log-loss = 71.54974698688189
Iteration 60. w = [0.87818506 2.25566788], log-loss = 71.54957352914379
Iteration 61. w = [0.87893301 2.25636008], log-loss = 71.5494240934032
Iteration 62. w = [0.87962717 2.25700294], log-loss = 71.54929533544599
Iteration 63. w = [0.88027145 2.25759998], log-loss = 71.54918438010375
Iteration 64. w = [0.88086948 2.25815449], log-loss = 71.54908875477132
Iteration 65. w = [0.88142462 2.25866951], log-loss = 71.54900633248657
Iteration 66. w = [0.88193997 2.25914785], log-loss = 71.5489352831695
Iteration 67. w = [0.88241841 2.25959214], log-loss = 71.54887403182899
Iteration 68. w = [0.88286261 2.26000481], log-loss = 71.54882122272359
Iteration 69. w = [0.88327503 2.26038811], log-loss = 71.54877568861369
Iteration 70. w = [0.88365798 2.26074415], log-loss = 71.54873642437141
Iteration 71. w = [0.88401356 2.26107486], log-loss = 71.54870256432105
Iteration 72. w = [0.88434375 2.26138204], log-loss = 71.54867336277557
Iteration 73. w = [0.88465038 2.26166739], log-loss = 71.54864817731249
Iteration 74. w = [0.88493512 2.26193244], log-loss = 71.5486264543998
Iteration 75. w = [0.88519956 2.26217866], log-loss = 71.54860771703694
Iteration 76. w = [0.88544515 2.26240737], log-loss = 71.54859155412592
Iteration 77. w = [0.88567323 2.26261983], log-loss = 71.5485776113283
Iteration 78. w = [0.88588507 2.2628172 ], log-loss = 71.54856558319858
Iteration 79. w = [0.88608182 2.26300053], log-loss = 71.54855520641219
Iteration 80. w = [0.88626455 2.26317085], log-loss = 71.5485462539389
Iteration 81. w = [0.88643428 2.26332906], log-loss = 71.5485385300238
Iteration 82. w = [0.88659193 2.26347604], log-loss = 71.54853186586675
Iteration 83. w = [0.88673837 2.26361258], log-loss = 71.54852611590074
Iteration 84. w = [0.88687438 2.26373942], log-loss = 71.5485211545856
Iteration 85. w = [0.88700072 2.26385725], log-loss = 71.54851687364747
Iteration 86. w = [0.88711808 2.26396671], log-loss = 71.54851317969764
Iteration 87. w = [0.88722709 2.26406841], log-loss = 71.54850999218289
Iteration 88. w = [0.88732836 2.26416288], log-loss = 71.5485072416178
Iteration 89. w = [0.88742242 2.26425064], log-loss = 71.5485048680613
Iteration 90. w = [0.8875098  2.26433218], log-loss = 71.5485028198036
Iteration 91. w = [0.88759098 2.26440792], log-loss = 71.54850105223437
Iteration 92. w = [0.88766638 2.26447829], log-loss = 71.54849952686651
Iteration 93. w = [0.88773643 2.26454366], log-loss = 71.5484982104944
Iteration 94. w = [0.8878015 2.2646044], log-loss = 71.5484970744685
Iteration 95. w = [0.88786194 2.26466082], log-loss = 71.54849609406955
Iteration 96. w = [0.8879181  2.26471324], log-loss = 71.54849524796849
Iteration 97. w = [0.88797027 2.26476193], log-loss = 71.54849451776158
Iteration 98. w = [0.88801873 2.26480718], log-loss = 71.54849388756827
Iteration 99. w = [0.88806375 2.26484921], log-loss = 71.54849334368532
Iteration 100. w = [0.88810557 2.26488825], log-loss = 71.54849287428814
```

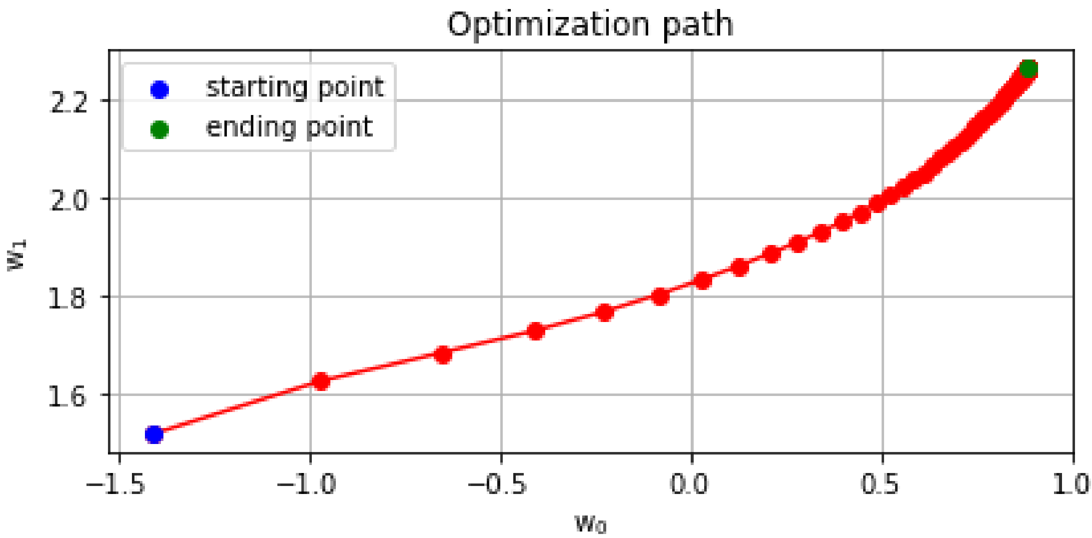
[31]:
```
# Plot the results
import matplotlib.pyplot as plt
old_ws = np.array(old_ws)
```

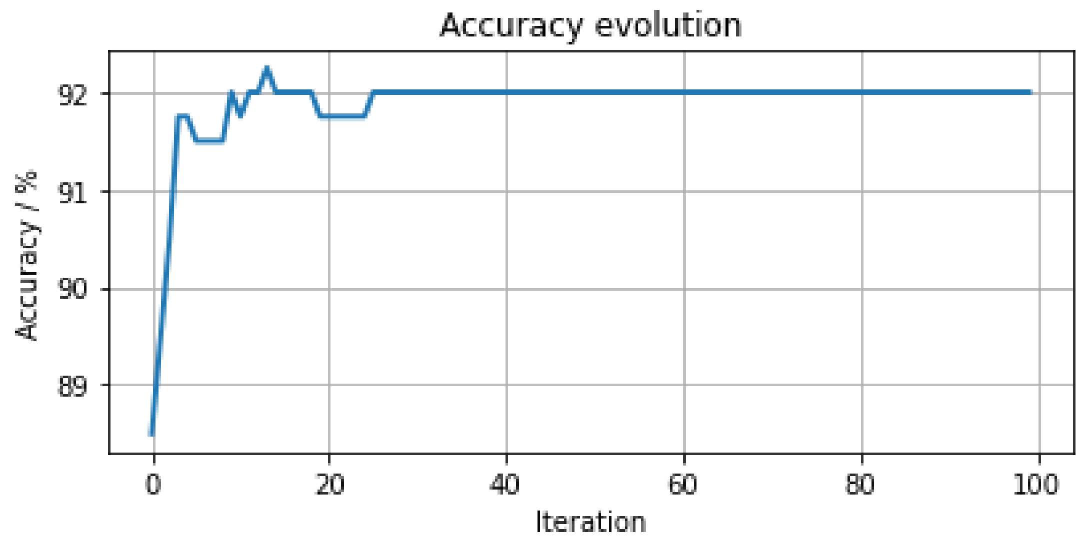
```
old_accs = np.array(old_accs)

fig = plt.figure(figsize = [6,6])
ax1 = fig.add_subplot(211)
ax1.plot(old_ws[:,0], old_ws[:,1], 'ro-')
blue_dot = ax1.scatter(old_ws[0,0], old_ws[0,1], color='blue', zorder=10)
green_dot = ax1.scatter(old_ws[-1,0], old_ws[-1,1], color='green', zorder=10)
plt.legend([blue_dot, green_dot], ['starting point','ending point'])
plt.grid()
plt.xlabel('w$_0$')
plt.ylabel('w$_1$')
plt.title('Optimization path')

plt.subplot(212)
plt.plot(100.0 * old_accs, linewidth = 2)
plt.grid()
plt.ylabel('Accuracy / %')
plt.xlabel('Iteration')
plt.title('Accuracy evolution')
plt.tight_layout()

plt.show()
```





5 Question 4

Answered to question b)

[32]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import MaxPooling2D

N = 32 # Number of feature maps
w, h = 5, 5 # Conv. window size
```

```
model = Sequential()
model.add(Conv2D(N, (w, h),input_shape=(64, 64, 3),activation = 'relu',padding␣
 ↪= 'same'))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(N, (w, h),activation = 'relu',padding = 'same'))
model.add(MaxPooling2D((4,4)))
model.add(Flatten())
model.add(Dense(100, activation = 'sigmoid'))
model.add(Dense(9, activation = 'softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 100)               51300
_________________________________________________________________
dense_1 (Dense)              (None, 9)                 909
=================================================================
Total params: 80,273
Trainable params: 80,273
Non-trainable params: 0
_________________________________________________________________
```

6 Question 5

[33]:
```
# a)
import numpy as np
import os
from skimage.io import imread_collection
from sklearn.model_selection import train_test_split
import cv2

# Compile the network of Question 4 b
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
# Load the data
os.chdir('/home/tuomas/Python/DATA.ML.200/Ex3')
images = []
labels = []
for i in range(0,9):
    fn = '0000{}/*.jpg'.format(i)
    print(fn)
    imgs = imread_collection(fn)
    images.append(np.array(imgs, dtype='object'))
    labels.append( np.ones(len(imgs)) * i )

images = np.concatenate(images)
labels = np.concatenate(labels).astype('uint8')
```

```
00000/*.jpg
00001/*.jpg
00002/*.jpg
00003/*.jpg
00004/*.jpg
00005/*.jpg
00006/*.jpg
00007/*.jpg
00008/*.jpg
```

[34]:
```
# Resize all images to 64 x 64
images_resized = []
for img in images:
    img_r = cv2.resize(img, (64,64))
    images_resized.append(img_r/255.)

images_resized = np.array(images_resized)
```

[35]:
```
# Create training and testing sets.
from tensorflow.keras.utils import to_categorical
trainX, testX, trainY, testY = train_test_split(images_resized,
                                                labels,
                                                test_size=0.15)

# One-hot encoding
trainY_cat = to_categorical(trainY, num_classes=9)
testY_cat = to_categorical(testY, num_classes=9)
```

[36]:
```
# b)
# Train the model
```

```python
history = model.fit(trainX, trainY_cat, batch_size=32, epochs=20,␣
↪validation_data=(testX, testY_cat), verbose=1)
# Evaluate the model
test_loss, test_acc = model.evaluate(testX, testY_cat, verbose=2)
print("Accuracy of CNN = {}".format(test_acc))
```

```
Epoch 1/20
237/237 [==============================] - 3s 14ms/step - loss: 1.8738 -
accuracy: 0.2727 - val_loss: 1.5935 - val_accuracy: 0.4256
Epoch 2/20
237/237 [==============================] - 1s 5ms/step - loss: 1.3079 -
accuracy: 0.5489 - val_loss: 0.9500 - val_accuracy: 0.7210
Epoch 3/20
237/237 [==============================] - 1s 5ms/step - loss: 0.7369 -
accuracy: 0.7832 - val_loss: 0.5339 - val_accuracy: 0.8504
Epoch 4/20
237/237 [==============================] - 1s 5ms/step - loss: 0.4071 -
accuracy: 0.8928 - val_loss: 0.3252 - val_accuracy: 0.9155
Epoch 5/20
237/237 [==============================] - 1s 5ms/step - loss: 0.2296 -
accuracy: 0.9519 - val_loss: 0.2471 - val_accuracy: 0.9342
Epoch 6/20
237/237 [==============================] - 1s 5ms/step - loss: 0.1483 -
accuracy: 0.9690 - val_loss: 0.1675 - val_accuracy: 0.9604
Epoch 7/20
237/237 [==============================] - 1s 5ms/step - loss: 0.1018 -
accuracy: 0.9819 - val_loss: 0.1215 - val_accuracy: 0.9678
Epoch 8/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0691 -
accuracy: 0.9886 - val_loss: 0.0948 - val_accuracy: 0.9723
Epoch 9/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0439 -
accuracy: 0.9963 - val_loss: 0.0811 - val_accuracy: 0.9798
Epoch 10/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0342 -
accuracy: 0.9967 - val_loss: 0.0707 - val_accuracy: 0.9828
Epoch 11/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0228 -
accuracy: 0.9987 - val_loss: 0.0669 - val_accuracy: 0.9791
Epoch 12/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0185 -
accuracy: 0.9991 - val_loss: 0.0664 - val_accuracy: 0.9813
Epoch 13/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0293 -
accuracy: 0.9955 - val_loss: 0.0688 - val_accuracy: 0.9776
Epoch 14/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0100 -
```

```
accuracy: 0.9997 - val_loss: 0.0473 - val_accuracy: 0.9843
Epoch 15/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0129 -
accuracy: 0.9989 - val_loss: 0.0790 - val_accuracy: 0.9768
Epoch 16/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0139 -
accuracy: 0.9984 - val_loss: 0.0565 - val_accuracy: 0.9843
Epoch 17/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0065 -
accuracy: 0.9997 - val_loss: 0.0435 - val_accuracy: 0.9865
Epoch 18/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0034 -
accuracy: 1.0000 - val_loss: 0.0362 - val_accuracy: 0.9895
Epoch 19/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0026 -
accuracy: 1.0000 - val_loss: 0.0354 - val_accuracy: 0.9873
Epoch 20/20
237/237 [==============================] - 1s 5ms/step - loss: 0.0020 -
accuracy: 1.0000 - val_loss: 0.0343 - val_accuracy: 0.9888
42/42 - 0s - loss: 0.0343 - accuracy: 0.9888
Accuracy of CNN = 0.9887808561325073
```

[ ]:

```
```