Computational Intelligence Lab

Assignment 2

Szymon Berdzik Inżynieria Oprogramowania

2.1 Load Fashion MNIST dataset

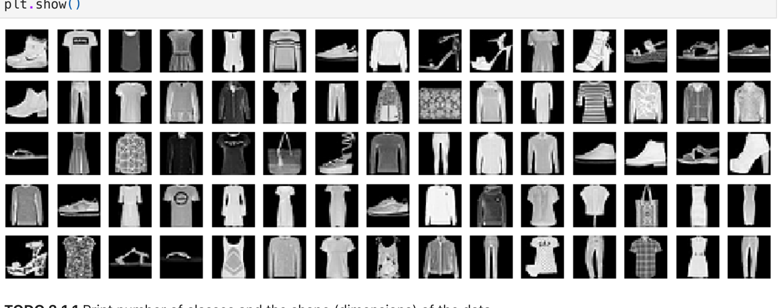
The following code loads Fashion MNIST dataset. More information about the dataset

We will concatnetate training and test sets to make an own split.

In [2]:
```
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X=np.concatenate((np.array(x_train),np.array(x_test)),axis=0);
y=np.concatenate((np.array(y_train),np.array(y_test)),axis=0);
# X=X.reshape(X.shape[0],X.shape[1]*X.shape[2])
X=X/255
```
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

Display a few dozen of images

In [3]:
```
import matplotlib.pyplot as plt
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
col2 = 15
row2 = 5
fig = plt.figure(figsize=(col2, row2))
for index2 in range(0, col2*row2):
    fig.add_subplot(row2, col2, index2 + 1)
    plt.axis('off')
    plt.imshow(X[index2]) # index of the sample image
plt.show()
```



TODO 2.1.1 Print number of classes and the shape (dimensions) of the data

In [4]:
```
nb_classes = len(np.unique(y))
shape = X.shape
```

In [5]:
```
print(f' #classes = {nb_classes} shape = {shape}')
```
```
 #classes = 10 shape = (70000, 28, 28)
```

Subset selection

We select a subset comprising only two classes, namely t-shirts/tops and trousers

In [6]:
```
X2 = X[(y==0) | (y==1)]
y2=y[(y==0) | (y==1)]
y2
```
Out[6]:
```
array([0, 0, 0, ..., 1, 1, 1], dtype=uint8)
```

TODO 2.1.2 Print number of classes and the shape (dimensions) of the data

In [7]:
```
print(f' #classes = {len(np.unique(y2))} shape = {y2.shape}')
```
```
 #classes = 2 shape = (14000,)
```

TODO 2.1.3 Display images from X2

In [8]:
```
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
col2 = 15
row2 = 5
fig = plt.figure(figsize=(col2, row2))
for index2 in range(0, col2*row2):
    fig.add_subplot(row2, col2, index2 + 1)
    plt.axis('off')
    plt.imshow(X2[index2]) # index of the sample image
plt.show()
```

Flatten images

In [9]:
```
X2=X2.reshape(X2.shape[0],-1)
X2
```
Out[9]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

TODO 2.1.4 Print the shape of X2 after flattening

In [10]:
```
print(f' #classes = {y2.max()+1}, shape = {X2.shape}')
```
```
 #classes = 2, shape = (14000, 784)
```

Train / test split

In [11]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.33, random_state=42)
```

2.2 Binary classification

- Build a model
- Compile (use binary_crossentropy as the loss function)
- Fit the training data, set epochs=10
- Display training history

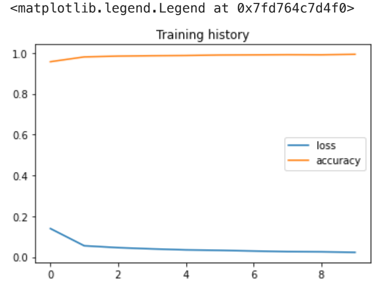
In [12]:
```
import tensorflow as tf
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                12560

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 12,705
Trainable params: 12,705
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
```

In [14]:
```
hist = model.fit(X_train,y_train,epochs=10,batch_size=128)
```
```
Epoch 1/10
74/74 [==============================] - 1s 3ms/step - loss: 0.1391 - accuracy: 0.9565
Epoch 2/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0547 - accuracy: 0.9801
Epoch 3/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0454 - accuracy: 0.9840
Epoch 4/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0394 - accuracy: 0.9858
Epoch 5/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0346 - accuracy: 0.9869
Epoch 6/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0323 - accuracy: 0.9893
Epoch 7/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0289 - accuracy: 0.9901
Epoch 8/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0259 - accuracy: 0.9909
Epoch 9/10
74/74 [==============================] - 0s 3ms/step - loss: 0.0251 - accuracy: 0.9904
Epoch 10/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0225 - accuracy: 0.9929
```

In [15]:
```
for k in hist.history:
  print(k)
```
```
loss
accuracy
```

In [16]:
```
plt.title('Training history')
for k in hist.history:
  plt.plot(hist.history[k],label=k)
plt.legend()
```
Out[16]:
```
<matplotlib.legend.Legend at 0x7fd764c7d4f0>
```



What is the performance of our classifier?

Testing on the training set

Collect the model predictions

In [17]:
```
probs = model.predict(X_train)
print(probs.shape)
probs
```
```
294/294 [==============================] - 0s 1ms/step
(9380, 1)
```
Out[17]:
```
array([[9.9965286e-01],
       [3.3811717e-10],
       [5.2905966e-06],
       ...,
       [9.0358907e-01],
       [9.6779895e-06],
       [3.5858875e-07]], dtype=float32)
```

We prefer a flat vector

In [18]:
```
# probs = model.predict(X_train)[:,0]
# probs
# or
probs=probs.flatten()
probs
```
Out[18]:
```
array([9.9965286e-01, 3.3811717e-10, 5.2905966e-06, ..., 9.0358907e-01,
       9.6779895e-06, 3.5858875e-07], dtype=float32)
```

Compute labels

In [19]:
```
y_pred = np.where(probs>.5,1,0)
y_pred
```
Out[19]:
```
array([1, 0, 0, ..., 1, 0, 0])
```

How to check results?

In [20]:
```
pairs = zip(y_train,y_pred)
for i,p in enumerate(pairs):
  if i>30:break
  print(p)
```
```
(1, 1)
(0, 0)
(0, 0)
(0, 0)
(1, 1)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(1, 1)
(1, 1)
(1, 1)
(0, 0)
(1, 1)
(0, 0)
(0, 0)
(1, 1)
(0, 0)
(1, 1)
(0, 0)
(1, 1)
(1, 0)
(1, 1)
(0, 0)
(0, 0)
(1, 1)
(1, 1)
(1, 1)
```

Or load it to Pandas, then it can be converted to a fully browseable data table (use the magic wand on the right)

In [21]:
```
import pandas as pd
df = pd.DataFrame(zip(y_train,y_pred))
df.head(df.size)
```
Out[21]:

	0	1
0	1	1
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
9375	1	1
9376	1	1
9377	1	1
9378	0	0
9379	0	0

9380 rows × 2 columns

However, we need a measure - a single number describing similarity of predictions and known labels

TODO 2.2.0 Compute this: $m = \frac{1}{n}\sum_{i=1}^{n} abs(y_{train}[i] - y_{pred}[i])$

In [22]:
```
def m(y, pred):
    n=len(y)
    return sum([ abs(y[i] - pred[i]) for i in range(0,n)])/n
```

In [23]:
```
print(f'm={m(y_train, y_pred)} 1-m={1-m(y_train, y_pred)}')
```
```
m=0.008422174840085287 1-m=0.9915778251599147
```

and compute accuracy. Compare results

In [24]:
```
from sklearn.metrics import accuracy_score
acc=accuracy_score(y_pred,y_train)
print(f'accuracy={acc}')
```
```
accuracy=0.9915778251599147
```

Compute the **confusion matrix**

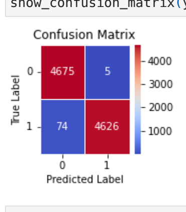
In [25]:
```
from sklearn.metrics import confusion_matrix, classification_report
matrix = confusion_matrix(y_train, y_pred)
matrix
```
Out[25]:
```
array([[4675,    5],
       [  74, 4626]])
```

Useful function (small adaptation)

In [26]:
```
import seaborn as sns
from sklearn import metrics

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

# Define the confusion matrix for the results
def show_confusion_matrix(validations, predictions, num_classes):
    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize=(num_classes, num_classes))
    hm = sns.heatmap(matrix,
                cmap='coolwarm',
                linecolor='white',
                linewidths=1,
                xticklabels=LABELS[0:num_classes],
                yticklabels=LABELS[0:num_classes],
                annot=True,
                fmt='d')
    plt.yticks(rotation = 0)  # Don't rotate (vertically) the y-axis labels
    # hm.set_ylim(0, len(matrix))
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

In [27]:
```
y_pred = np.where(probs>.5,1,0)
y_pred
```
Out[27]:
```
array([1, 0, 0, ..., 1, 0, 0])
```

In [28]:
```
show_confusion_matrix(y_train, y_pred, 2)
```



In [29]:
```
print(classification_report(y_train, y_pred))
```
```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      4680
           1       1.00      0.98      0.99      4700

    accuracy                           0.99      9380
   macro avg       0.99      0.99      0.99      9380
weighted avg       0.99      0.99      0.99      9380
```

Exact formuls for classification scores are given here (in Polish) silde 14 and following

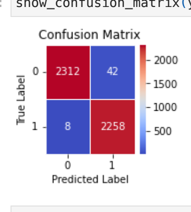
Validation on the test set

TODO 2.2.1 Repat the steps above on the test set. Replace X_train by X_test, etc.

In [30]:
```
probs = model.predict(X_test)
y_pred = np.where(probs > .5, 1, 0)
m_result = m(y_test, y_pred)
print(f'm={m_result} 1-m={1-m_result}')
```
```
145/145 [==============================] - 0s 1ms/step
m=[0.01082251] 1-m=[0.98917749]
```

In [31]:
```
acc=accuracy_score(y_pred, y_test)
print(f'accuracy={acc}')
```
```
accuracy=0.9891774891774892
```

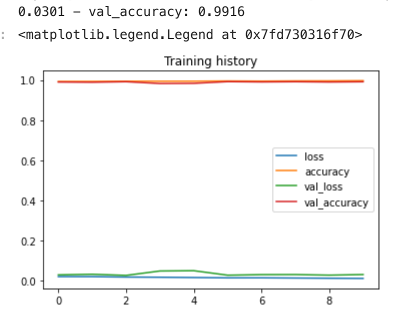
In [32]:
```
show_confusion_matrix(y_pred, y_test,2)
```



In [33]:
```
print(classification_report(y_test, y_pred))
```
```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      2320
           1       1.00      0.98      0.99      2300

    accuracy                           0.99      4620
   macro avg       0.99      0.99      0.99      4620
weighted avg       0.99      0.99      0.99      4620
```

TODO 2.2.2 supply X_test, y_test as validation data, fit the model and display plots

In [34]:
```
hist = model.fit(X_train,y_train,epochs=10,batch_size=128,validation_data=(X_test, y_test))
plt.title('Training history')
for k in hist.history:
  plt.plot(hist.history[k],label=k)
plt.legend()
```
```
Epoch 1/10
74/74 [==============================] - 0s 6ms/step - loss: 0.0198 - accuracy: 0.9931 - val_loss: 0.0286 - val_accuracy: 0.9903
Epoch 2/10
74/74 [==============================] - 0s 4ms/step - loss: 0.0197 - accuracy: 0.9934 - val_loss: 0.0316 - val_accuracy: 0.9892
Epoch 3/10
74/74 [==============================] - 0s 4ms/step - loss: 0.0177 - accuracy: 0.9942 - val_loss: 0.0260 - val_accuracy: 0.9918
Epoch 4/10
74/74 [==============================] - 0s 3ms/step - loss: 0.0167 - accuracy: 0.9943 - val_loss: 0.0479 - val_accuracy: 0.9829
Epoch 5/10
74/74 [==============================] - 0s 3ms/step - loss: 0.0153 - accuracy: 0.9945 - val_loss: 0.0493 - val_accuracy: 0.9838
Epoch 6/10
74/74 [==============================] - 0s 3ms/step - loss: 0.0137 - accuracy: 0.9955 - val_loss: 0.0270 - val_accuracy: 0.9922
Epoch 7/10
74/74 [==============================] - 0s 4ms/step - loss: 0.0139 - accuracy: 0.9948 - val_loss: 0.0297 - val_accuracy: 0.9907
Epoch 8/10
74/74 [==============================] - 0s 4ms/step - loss: 0.0128 - accuracy: 0.9958 - val_loss: 0.0301 - val_accuracy: 0.9918
Epoch 9/10
74/74 [==============================] - 0s 3ms/step - loss: 0.0116 - accuracy: 0.9961 - val_loss: 0.0272 - val_accuracy: 0.9903
Epoch 10/10
74/74 [==============================] - 0s 3ms/step - loss: 0.0108 - accuracy: 0.9970 - val_loss: 0.0301 - val_accuracy: 0.9916
```
Out[34]:
```
<matplotlib.legend.Legend at 0x7fd730316f70>
```



2.3 Binary classification on brest cancer dataset

The information on the dataset can be found here

In [35]:
```
from sklearn.datasets import load_breast_cancer

X,y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

TODO 2.3.1 Repeat the previous steps for to process the loaded data

- Create a model
- fit it setting epochs=100 and supplying validation data
- plot training history
- obtain predictions
- display confussion matrix
- and print the classification report

Caveat we seed everything possible to create reproducible results

In [36]:
```
import os
import random
def set_seeds(seed=1):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)

set_seeds(42)
```

In [37]:
```
import tensorflow as tf
from keras import models
from keras import layers
tf.random.set_seed(1)

model = models.Sequential()
model.add(layers.Dense(8, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

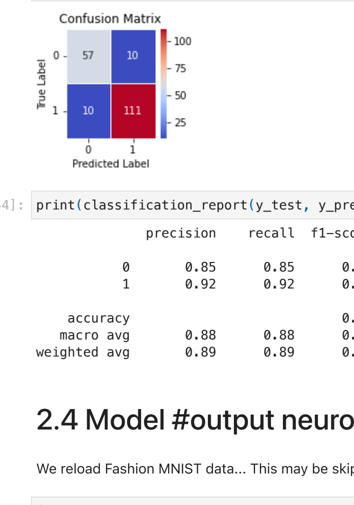
In [38]:
```
# use loss='binary_crossentropy'
model.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
```

In [39]:
```
# epochs=100, batch_size=y_train.shape[0], use validation data
hist = model.fit(X_train, y_train, epochs=100, batch_size=y_train.shape[0], validation_data=(X_test
```

```
Epoch 1/100
1/1 [==============================] - 1s 664ms/step - loss: 1.1419 - accuracy: 0.7638 - val_loss:
0.6137 - val_accuracy: 0.8138
Epoch 2/100
1/1 [==============================] - 0s 48ms/step - loss: 0.9202 - accuracy: 0.7612 - val_loss:
0.5920 - val_accuracy: 0.8032
Epoch 3/100
1/1 [==============================] - 0s 36ms/step - loss: 0.8497 - accuracy: 0.7507 - val_loss:
0.5896 - val_accuracy: 0.7766
Epoch 4/100
1/1 [==============================] - 0s 32ms/step - loss: 0.8120 - accuracy: 0.7244 - val_loss:
0.5941 - val_accuracy: 0.7606
Epoch 5/100
1/1 [==============================] - 0s 34ms/step - loss: 0.7843 - accuracy: 0.7060 - val_loss:
0.5981 - val_accuracy: 0.7181
Epoch 6/100
1/1 [==============================] - 0s 37ms/step - loss: 0.7650 - accuracy: 0.7034 - val_loss:
0.6022 - val_accuracy: 0.7128
Epoch 7/100
1/1 [==============================] - 0s 32ms/step - loss: 0.7524 - accuracy: 0.6903 - val_loss:
0.6059 - val_accuracy: 0.6968
Epoch 8/100
1/1 [==============================] - 0s 34ms/step - loss: 0.7423 - accuracy: 0.6850 - val_loss:
0.6089 - val_accuracy: 0.6968
Epoch 9/100
1/1 [==============================] - 0s 35ms/step - loss: 0.7337 - accuracy: 0.6745 - val_loss:
0.6122 - val_accuracy: 0.6862
Epoch 10/100
1/1 [==============================] - 0s 32ms/step - loss: 0.7279 - accuracy: 0.6745 - val_loss:
0.6145 - val_accuracy: 0.6755
Epoch 11/100
1/1 [==============================] - 0s 33ms/step - loss: 0.7246 - accuracy: 0.6693 - val_loss:
0.6164 - val_accuracy: 0.6809
Epoch 12/100
1/1 [==============================] - 0s 34ms/step - loss: 0.7216 - accuracy: 0.6667 - val_loss:
0.6181 - val_accuracy: 0.6755
Epoch 13/100
1/1 [==============================] - 0s 36ms/step - loss: 0.7189 - accuracy: 0.6667 - val_loss:
0.6196 - val_accuracy: 0.6702
Epoch 14/100
1/1 [==============================] - 0s 31ms/step - loss: 0.7165 - accuracy: 0.6614 - val_loss:
0.6200 - val_accuracy: 0.6755
Epoch 15/100
1/1 [==============================] - 0s 31ms/step - loss: 0.7143 - accuracy: 0.6640 - val_loss:
0.6205 - val_accuracy: 0.6755
Epoch 16/100
1/1 [==============================] - 0s 50ms/step - loss: 0.7121 - accuracy: 0.6667 - val_loss:
0.6206 - val_accuracy: 0.6755
Epoch 17/100
1/1 [==============================] - 0s 35ms/step - loss: 0.7100 - accuracy: 0.6640 - val_loss:
0.6206 - val_accuracy: 0.6436
Epoch 18/100
1/1 [==============================] - 0s 34ms/step - loss: 0.7079 - accuracy: 0.6194 - val_loss:
0.6202 - val_accuracy: 0.6436
Epoch 19/100
1/1 [==============================] - 0s 40ms/step - loss: 0.7058 - accuracy: 0.6194 - val_loss:
0.6197 - val_accuracy: 0.6436
Epoch 20/100
1/1 [==============================] - 0s 49ms/step - loss: 0.7037 - accuracy: 0.6194 - val_loss:
0.6192 - val_accuracy: 0.6436
Epoch 21/100
1/1 [==============================] - 0s 34ms/step - loss: 0.7016 - accuracy: 0.6194 - val_loss:
0.6186 - val_accuracy: 0.6436
Epoch 22/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6995 - accuracy: 0.6194 - val_loss:
0.6181 - val_accuracy: 0.6436
Epoch 23/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6974 - accuracy: 0.6194 - val_loss:
0.6176 - val_accuracy: 0.6436
Epoch 24/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6954 - accuracy: 0.6194 - val_loss:
0.6175 - val_accuracy: 0.6436
Epoch 25/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6933 - accuracy: 0.6194 - val_loss:
0.6166 - val_accuracy: 0.6436
Epoch 26/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6913 - accuracy: 0.6194 - val_loss:
0.6159 - val_accuracy: 0.6436
Epoch 27/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6892 - accuracy: 0.6194 - val_loss:
0.6145 - val_accuracy: 0.6436
Epoch 28/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6871 - accuracy: 0.6194 - val_loss:
0.6116 - val_accuracy: 0.6436
Epoch 29/100
1/1 [==============================] - 0s 62ms/step - loss: 0.6846 - accuracy: 0.6194 - val_loss:
0.6096 - val_accuracy: 0.6436
Epoch 30/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6823 - accuracy: 0.6194 - val_loss:
0.6086 - val_accuracy: 0.6436
Epoch 31/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6801 - accuracy: 0.6194 - val_loss:
0.6075 - val_accuracy: 0.6436
Epoch 32/100
1/1 [==============================] - 0s 37ms/step - loss: 0.6779 - accuracy: 0.6194 - val_loss:
0.6068 - val_accuracy: 0.6436
Epoch 33/100
1/1 [==============================] - 0s 30ms/step - loss: 0.6758 - accuracy: 0.6194 - val_loss:
0.6046 - val_accuracy: 0.6436
Epoch 34/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6736 - accuracy: 0.6194 - val_loss:
0.6046 - val_accuracy: 0.6436
Epoch 35/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6716 - accuracy: 0.6194 - val_loss:
0.6019 - val_accuracy: 0.6436
Epoch 36/100
1/1 [==============================] - 0s 50ms/step - loss: 0.6696 - accuracy: 0.6194 - val_loss:
0.6019 - val_accuracy: 0.6436
Epoch 37/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6677 - accuracy: 0.6194 - val_loss:
0.6026 - val_accuracy: 0.6436
Epoch 38/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6658 - accuracy: 0.6194 - val_loss:
0.6057 - val_accuracy: 0.6436
Epoch 39/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6644 - accuracy: 0.6194 - val_loss:
0.6035 - val_accuracy: 0.6436
Epoch 40/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6626 - accuracy: 0.6194 - val_loss:
0.6015 - val_accuracy: 0.6436
Epoch 41/100
1/1 [==============================] - 0s 37ms/step - loss: 0.6609 - accuracy: 0.6194 - val_loss:
0.5950 - val_accuracy: 0.6436
Epoch 42/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6590 - accuracy: 0.6194 - val_loss:
0.5948 - val_accuracy: 0.6436
Epoch 43/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6574 - accuracy: 0.6194 - val_loss:
0.5959 - val_accuracy: 0.6436
Epoch 44/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6559 - accuracy: 0.6194 - val_loss:
0.5961 - val_accuracy: 0.6436
Epoch 45/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6544 - accuracy: 0.6194 - val_loss:
0.5931 - val_accuracy: 0.6436
Epoch 46/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6528 - accuracy: 0.6194 - val_loss:
0.5933 - val_accuracy: 0.6436
Epoch 47/100
1/1 [==============================] - 0s 38ms/step - loss: 0.6515 - accuracy: 0.6194 - val_loss:
0.5930 - val_accuracy: 0.6436
Epoch 48/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6503 - accuracy: 0.6194 - val_loss:
0.5896 - val_accuracy: 0.6436
Epoch 49/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6490 - accuracy: 0.6194 - val_loss:
0.5909 - val_accuracy: 0.6436
Epoch 50/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6478 - accuracy: 0.6194 - val_loss:
0.5911 - val_accuracy: 0.6436
Epoch 51/100
1/1 [==============================] - 0s 35ms/step - loss: 0.6467 - accuracy: 0.6194 - val_loss:
0.5863 - val_accuracy: 0.6436
Epoch 52/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6452 - accuracy: 0.6194 - val_loss:
0.5839 - val_accuracy: 0.6436
Epoch 53/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6441 - accuracy: 0.6194 - val_loss:
0.5867 - val_accuracy: 0.6436
Epoch 54/100
1/1 [==============================] - 0s 35ms/step - loss: 0.6428 - accuracy: 0.6194 - val_loss:
0.5900 - val_accuracy: 0.6436
Epoch 55/100
1/1 [==============================] - 0s 47ms/step - loss: 0.6422 - accuracy: 0.6194 - val_loss:
0.5898 - val_accuracy: 0.6436
Epoch 56/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6414 - accuracy: 0.6194 - val_loss:
0.5806 - val_accuracy: 0.6436
Epoch 57/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6400 - accuracy: 0.6194 - val_loss:
0.5820 - val_accuracy: 0.6436
Epoch 58/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6388 - accuracy: 0.6194 - val_loss:
0.5794 - val_accuracy: 0.6436
Epoch 59/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6379 - accuracy: 0.6194 - val_loss:
0.5818 - val_accuracy: 0.6436
Epoch 60/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6368 - accuracy: 0.6194 - val_loss:
0.5811 - val_accuracy: 0.6436
Epoch 61/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6359 - accuracy: 0.6194 - val_loss:
0.5775 - val_accuracy: 0.6436
Epoch 62/100
1/1 [==============================] - 0s 39ms/step - loss: 0.6349 - accuracy: 0.6194 - val_loss:
0.5794 - val_accuracy: 0.6436
Epoch 63/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6340 - accuracy: 0.6194 - val_loss:
0.5731 - val_accuracy: 0.6436
Epoch 64/100
1/1 [==============================] - 0s 39ms/step - loss: 0.6332 - accuracy: 0.6194 - val_loss:
0.5755 - val_accuracy: 0.6436
Epoch 65/100
1/1 [==============================] - 0s 35ms/step - loss: 0.6322 - accuracy: 0.6194 - val_loss:
0.5764 - val_accuracy: 0.6436
Epoch 66/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6312 - accuracy: 0.6194 - val_loss:
0.5716 - val_accuracy: 0.6436
Epoch 67/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6303 - accuracy: 0.6194 - val_loss:
0.5689 - val_accuracy: 0.6436
Epoch 68/100
1/1 [==============================] - 0s 44ms/step - loss: 0.6292 - accuracy: 0.6194 - val_loss:
0.5815 - val_accuracy: 0.6436
Epoch 69/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6292 - accuracy: 0.6194 - val_loss:
0.5675 - val_accuracy: 0.6436
Epoch 70/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6274 - accuracy: 0.6194 - val_loss:
0.5685 - val_accuracy: 0.6436
Epoch 71/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6264 - accuracy: 0.6194 - val_loss:
0.5670 - val_accuracy: 0.6436
Epoch 72/100
1/1 [==============================] - 0s 35ms/step - loss: 0.6253 - accuracy: 0.6194 - val_loss:
0.5665 - val_accuracy: 0.6436
Epoch 73/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6242 - accuracy: 0.6194 - val_loss:
0.5625 - val_accuracy: 0.6436
Epoch 74/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6233 - accuracy: 0.6194 - val_loss:
0.5665 - val_accuracy: 0.6436
Epoch 75/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6223 - accuracy: 0.6194 - val_loss:
0.5602 - val_accuracy: 0.6436
Epoch 76/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6210 - accuracy: 0.6194 - val_loss:
0.5627 - val_accuracy: 0.6436
Epoch 77/100
1/1 [==============================] - 0s 35ms/step - loss: 0.6199 - accuracy: 0.6194 - val_loss:
0.5555 - val_accuracy: 0.6436
Epoch 78/100
1/1 [==============================] - 0s 32ms/step - loss: 0.6188 - accuracy: 0.6194 - val_loss:
0.5720 - val_accuracy: 0.6436
Epoch 79/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6187 - accuracy: 0.6194 - val_loss:
0.5560 - val_accuracy: 0.6436
Epoch 80/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6163 - accuracy: 0.6194 - val_loss:
0.5522 - val_accuracy: 0.6436
Epoch 81/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6150 - accuracy: 0.6194 - val_loss:
0.5537 - val_accuracy: 0.6436
Epoch 82/100
1/1 [==============================] - 0s 42ms/step - loss: 0.6138 - accuracy: 0.6194 - val_loss:
0.5443 - val_accuracy: 0.6436
Epoch 83/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6132 - accuracy: 0.6194 - val_loss:
0.5491 - val_accuracy: 0.6436
Epoch 84/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6115 - accuracy: 0.6194 - val_loss:
0.5511 - val_accuracy: 0.6436
Epoch 85/100
1/1 [==============================] - 0s 31ms/step - loss: 0.6101 - accuracy: 0.6194 - val_loss:
0.5499 - val_accuracy: 0.6436
Epoch 86/100
1/1 [==============================] - 0s 35ms/step - loss: 0.6085 - accuracy: 0.6194 - val_loss:
0.5461 - val_accuracy: 0.6436
Epoch 87/100
1/1 [==============================] - 0s 37ms/step - loss: 0.6070 - accuracy: 0.6194 - val_loss:
0.5525 - val_accuracy: 0.6436
Epoch 88/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6070 - accuracy: 0.6194 - val_loss:
0.5381 - val_accuracy: 0.6436
Epoch 89/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6047 - accuracy: 0.6194 - val_loss:
0.5518 - val_accuracy: 0.6436
Epoch 90/100
1/1 [==============================] - 0s 39ms/step - loss: 0.6050 - accuracy: 0.6194 - val_loss:
0.5286 - val_accuracy: 0.6436
Epoch 91/100
1/1 [==============================] - 0s 34ms/step - loss: 0.6035 - accuracy: 0.6194 - val_loss:
0.5520 - val_accuracy: 0.6436
Epoch 92/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6034 - accuracy: 0.6194 - val_loss:
0.5323 - val_accuracy: 0.6436
Epoch 93/100
1/1 [==============================] - 0s 34ms/step - loss: 0.5998 - accuracy: 0.6194 - val_loss:
0.5414 - val_accuracy: 0.8564
Epoch 94/100
1/1 [==============================] - 0s 35ms/step - loss: 0.5991 - accuracy: 0.8346 - val_loss:
0.5281 - val_accuracy: 0.8830
Epoch 95/100
1/1 [==============================] - 0s 37ms/step - loss: 0.5974 - accuracy: 0.8346 - val_loss:
0.5316 - val_accuracy: 0.8830
Epoch 96/100
1/1 [==============================] - 0s 51ms/step - loss: 0.5963 - accuracy: 0.8320 - val_loss:
0.5252 - val_accuracy: 0.8883
Epoch 97/100
1/1 [==============================] - 0s 40ms/step - loss: 0.5954 - accuracy: 0.8346 - val_loss:
0.5275 - val_accuracy: 0.8883
Epoch 98/100
1/1 [==============================] - 0s 38ms/step - loss: 0.5939 - accuracy: 0.8373 - val_loss:
0.5251 - val_accuracy: 0.8883
Epoch 99/100
1/1 [==============================] - 0s 36ms/step - loss: 0.5929 - accuracy: 0.8373 - val_loss:
0.5271 - val_accuracy: 0.8830
Epoch 100/100
1/1 [==============================] - 0s 35ms/step - loss: 0.5917 - accuracy: 0.8346 - val_loss:
0.5220 - val_accuracy: 0.8936
```

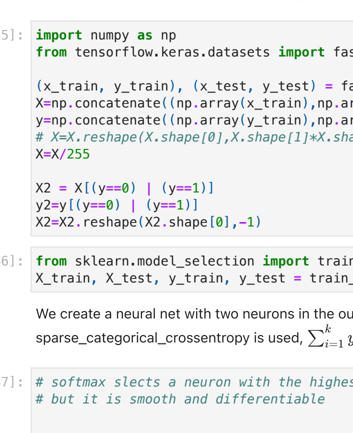
In [40]:
```python
plt.plot(hist.history['loss'],label='loss')
plt.plot(hist.history['val_loss'],label='val_loss')
plt.legend()
```

Out[40]: <matplotlib.legend.Legend at 0x7fd75e07cfa0>



In [41]:
```python
plt.plot(hist.history['accuracy'],label='acc')
plt.plot(hist.history['val_accuracy'],label='val_acc')
plt.legend()
```

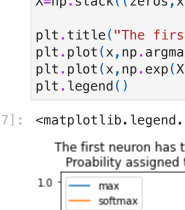
Out[41]: <matplotlib.legend.Legend at 0x7fd75e04c8b0>



In [42]:
```python
probs = model.predict(X_test).flatten()
y_pred = np.where(probs > .5, 1, 0)
```

```
6/6 [==============================] - 0s 2ms/step
```

In [43]:
```python
show_confusion_matrix(y_test, y_pred, 2)
```



In [44]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        67
           1       0.92      0.92      0.92       121

    accuracy                           0.89       188
   macro avg       0.88      0.88      0.88       188
weighted avg       0.89      0.89      0.89       188
```

2.4 Model #output neurons = #classes

We reload Fashion MNIST data... This may be skipped

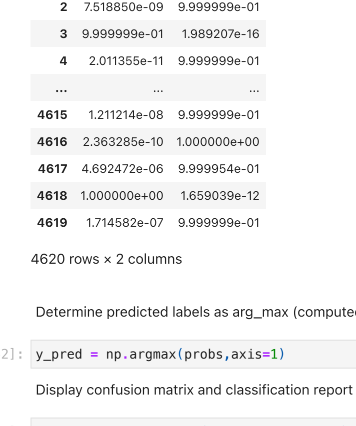
In [45]:
```python
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X=np.concatenate((np.array(x_train),np.array(x_test)),axis=0);
y=np.concatenate((np.array(y_train),np.array(y_test)),axis=0);
# X=X.reshape(X.shape[0],X.shape[1]*X.shape[2])
X=X/255

X2 = X[(y==0) | (y==1)]
y2=y[(y==0) | (y==1)]
X2=X2.reshape(X2.shape[0],-1)
```

In [46]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.33, random_state=42)
```

We create a neural net with two neurons in the output layer and softmax activation function. As the loss, sparse_categorical_crossentropy is used, $\sum_{i=1}^{k} y_i \cdot ln(p_i)$

In [47]:
```python
# softmax slects a neuron with the highest probability value
# but it is smooth and differentiable


x=np.linspace(-20,20,200)
zeros = x*0
X=np.stack((zeros,x),axis=-1)

plt.title("The first neuron has the (constant) value 0, the second x.\nProability assigned to the s
plt.plot(x,np.argmax(X,axis=1),label='max')
plt.plot(x,np.exp(X[:,1])/((np.exp(X[:,0])+np.exp(X[:,1]))),label='softmax')
plt.legend()
```

Out[47]: <matplotlib.legend.Legend at 0x7fd75dfab1f0>



In [48]:
```python
from keras import models
from keras import layers


num_classes = y_train.max()+1

network = models.Sequential()
network.add(layers.Dense(256, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(40, activation='relu'))
network.add(layers.Dense(num_classes, activation='softmax'))
network.compile(optimizer='rmsprop',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

In [49]:
```python
hist = network.fit(X_train,y_train,epochs=10,batch_size=128)
```

```
Epoch 1/10
74/74 [==============================] - 1s 5ms/step - loss: 0.1136 - accuracy: 0.9539
Epoch 2/10
74/74 [==============================] - 0s 5ms/step - loss: 0.0517 - accuracy: 0.9834
Epoch 3/10
74/74 [==============================] - 0s 5ms/step - loss: 0.0359 - accuracy: 0.9874
Epoch 4/10
74/74 [==============================] - 0s 5ms/step - loss: 0.0290 - accuracy: 0.9900
Epoch 5/10
74/74 [==============================] - 0s 5ms/step - loss: 0.0243 - accuracy: 0.9916
Epoch 6/10
74/74 [==============================] - 0s 5ms/step - loss: 0.0241 - accuracy: 0.9910
Epoch 7/10
74/74 [==============================] - 1s 8ms/step - loss: 0.0221 - accuracy: 0.9936
Epoch 8/10
74/74 [==============================] - 1s 8ms/step - loss: 0.0187 - accuracy: 0.9933
Epoch 9/10
74/74 [==============================] - 1s 8ms/step - loss: 0.0177 - accuracy: 0.9936
Epoch 10/10
74/74 [==============================] - 1s 8ms/step - loss: 0.0110 - accuracy: 0.9952
```

Get output probabilities

In [50]:
```python
probs = network.predict(X_test)
```

```
145/145 [==============================] - 0s 2ms/step
```

And load them to a data frame

In [51]:
```python
import pandas as pd

df = pd.DataFrame(probs)
df.head(df.size)
```

Out[51]:

	0	1
0	9.999999e-01	5.036000e-10
1	9.999999e-01	2.522523e-09
2	7.518850e-09	9.999999e-01
3	9.999999e-01	1.989207e-16
4	2.011355e-11	9.999999e-01
...	...	...
4615	1.211214e-08	9.999999e-01
4616	2.363285e-10	1.000000e+00
4617	4.692472e-06	9.999954e-01
4618	1.000000e+00	1.659039e-12
4619	1.714582e-07	9.999999e-01

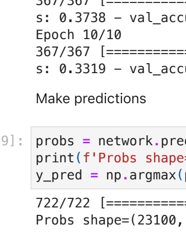
4620 rows × 2 columns

Determine predicted labels as arg_max (computed horizontally)

In [52]:
```python
y_pred = np.argmax(probs,axis=1)
```

Display confusion matrix and classification report

In [53]:
```python
show_confusion_matrix(y_test, y_pred, 2)
```



In [54]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2320
           1       0.99      0.99      0.99      2300

    accuracy                           0.99      4620
   macro avg       0.99      0.99      0.99      4620
weighted avg       0.99      0.99      0.99      4620
```

2.5 Build a model for all ten fashion classes

In [55]:
```python
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X=np.concatenate((np.array(x_train),np.array(x_test)),axis=0);
y=np.concatenate((np.array(y_train),np.array(y_test)),axis=0);

X=X/255

X=X.reshape(X.shape[0],-1)
```

In [56]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

TODO 2.5.1 Use the same model configuration, but adapt it to appropriate number of classes

In [57]:
```python
from keras import models
from keras import layers


num_classes = len(np.unique(y))
num_classes
# create a model add layers, compile

network = models.Sequential()
network.add(layers.Dense(256, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(40, activation='relu'))
network.add(layers.Dense(num_classes, activation='softmax'))
network.compile(optimizer='rmsprop',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Fit the model

In [58]:
```python
hist = network.fit(X_train,y_train,epochs=10,batch_size=128, validation_data=(X_test, y_test))
```

```
Epoch 1/10
367/367 [==============================] - 3s 7ms/step - loss: 0.6095 - accuracy: 0.7824 - val_los
s: 0.5265 - val_accuracy: 0.8079
Epoch 2/10
367/367 [==============================] - 2s 7ms/step - loss: 0.4170 - accuracy: 0.8475 - val_los
s: 0.4623 - val_accuracy: 0.8336
Epoch 3/10
367/367 [==============================] - 3s 8ms/step - loss: 0.3719 - accuracy: 0.8639 - val_los
s: 0.4036 - val_accuracy: 0.8477
Epoch 4/10
367/367 [==============================] - 4s 12ms/step - loss: 0.3434 - accuracy: 0.8745 - val_lo
ss: 0.3673 - val_accuracy: 0.8641
Epoch 5/10
367/367 [==============================] - 2s 6ms/step - loss: 0.3241 - accuracy: 0.8818 - val_los
s: 0.3703 - val_accuracy: 0.8665
Epoch 6/10
367/367 [==============================] - 2s 6ms/step - loss: 0.3063 - accuracy: 0.8863 - val_los
s: 0.3604 - val_accuracy: 0.8703
Epoch 7/10
367/367 [==============================] - 2s 7ms/step - loss: 0.2906 - accuracy: 0.8918 - val_los
s: 0.4169 - val_accuracy: 0.8574
Epoch 8/10
367/367 [==============================] - 2s 6ms/step - loss: 0.2782 - accuracy: 0.8959 - val_los
s: 0.3540 - val_accuracy: 0.8759
Epoch 9/10
367/367 [==============================] - 3s 9ms/step - loss: 0.2673 - accuracy: 0.9016 - val_los
s: 0.3738 - val_accuracy: 0.8649
Epoch 10/10
367/367 [==============================] - 2s 7ms/step - loss: 0.2556 - accuracy: 0.9043 - val_los
s: 0.3319 - val_accuracy: 0.8806
```

Make predictions

In [59]:
```python
probs = network.predict(X_test)
print(f'Probs shape={probs.shape}')
y_pred = np.argmax(probs,axis=1)
```

```
722/722 [==============================] - 1s 2ms/step
Probs shape=(23100, 10)
```

Show confusion matrix and scores (classification report)

In [60]:
```python
show_confusion_matrix(y_test, y_pred, num_classes)
```


True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1907	0	25	57	4	1	297	0	8	0
1	6	2293	5	24	6	1	9	0	1	0
2	58	1	1910	15	123	0	176	0	4	0
3	58	32	28	2027	134	0	60	1	2	0
4	3	3	324	34	1705	0	217	0	3	0
5	0	0	1	3	0	2287	0	39	3	15
6	305	3	201	33	100	0	1676	0	7	0
7	0	0	0	0	0	47	0	2206	5	29
8	11	2	12	5	8	9	32	10	2133	2
9	0	0	0	0	0	19	0	134	2	2199

In [61]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      2299
           1       0.98      0.98      0.98      2345
           2       0.76      0.84      0.80      2287
           3       0.92      0.87      0.89      2342
           4       0.82      0.74      0.78      2289
           5       0.97      0.97      0.97      2348
           6       0.68      0.72      0.70      2325
           7       0.92      0.96      0.94      2287
           8       0.98      0.96      0.97      2224
           9       0.98      0.93      0.96      2354

    accuracy                           0.88     23100
   macro avg       0.88      0.88      0.88     23100
weighted avg       0.88      0.88      0.88     23100
```

TODO 2.5.2 Analyze the results. Which fashion classes are wrongly classified. Can you explain that by similarity of forms?

2.6 Analyze the iris dataset

You can find the dataset description here

TODO 2.6.1 Implement the following steps

- First load data (code provided)
- Create a neural network comprising one hidden layer with 4 units
- Experimentaly establish the number of epochs during training.
- Provide validation data
- Display loss/validation loss and accuracies
- Predict output labels
- Display the confussion matrix and scores

In [62]:
```python
from sklearn.datasets import load_iris

X,y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [89]:
```python
# Build the model

from keras import models
from keras import layers

num_classes = len(np.unique(y))

network = models.Sequential()
network.add(layers.Dense(256, activation='relu', input_shape=(X.shape[1],)))
network.add(layers.Dense(40, activation='relu'))
network.add(layers.Dense(num_classes, activation='softmax'))
network.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

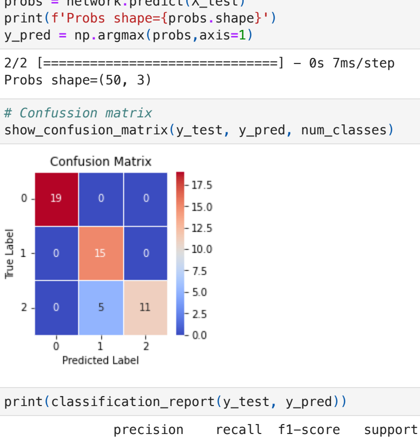
In [90]:
```python
#train

hist =  network.fit(X_train, y_train, epochs=100, batch_size=128, validation_data=(X_test, y_test))
```

```
Epoch 1/100
1/1 [==============================] - 1s 1s/step - loss: 1.1063 - accuracy: 0.3400 - val_loss: 0.
9881 - val_accuracy: 0.3200
Epoch 2/100
1/1 [==============================] - 0s 69ms/step - loss: 0.9608 - accuracy: 0.3500 - val_loss:
0.8713 - val_accuracy: 0.9600
Epoch 3/100
1/1 [==============================] - 0s 33ms/step - loss: 0.8816 - accuracy: 0.9400 - val_loss:
0.8225 - val_accuracy: 0.7000
Epoch 4/100
1/1 [==============================] - 0s 34ms/step - loss: 0.8341 - accuracy: 0.6500 - val_loss:
0.7728 - val_accuracy: 0.8600
Epoch 5/100
1/1 [==============================] - 0s 31ms/step - loss: 0.7926 - accuracy: 0.8500 - val_loss:
0.7204 - val_accuracy: 0.7000
Epoch 6/100
1/1 [==============================] - 0s 35ms/step - loss: 0.7446 - accuracy: 0.6500 - val_loss:
0.6820 - val_accuracy: 0.8800
Epoch 7/100
1/1 [==============================] - 0s 36ms/step - loss: 0.7110 - accuracy: 0.9000 - val_loss:
0.6490 - val_accuracy: 0.7000
Epoch 8/100
1/1 [==============================] - 0s 33ms/step - loss: 0.6814 - accuracy: 0.6700 - val_loss:
0.6214 - val_accuracy: 0.9000
Epoch 9/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6549 - accuracy: 0.9100 - val_loss:
0.5972 - val_accuracy: 0.7800
Epoch 10/100
1/1 [==============================] - 0s 36ms/step - loss: 0.6329 - accuracy: 0.7000 - val_loss:
0.5758 - val_accuracy: 0.9800
Epoch 11/100
1/1 [==============================] - 0s 40ms/step - loss: 0.6120 - accuracy: 0.9400 - val_loss:
0.5514 - val_accuracy: 0.7600
Epoch 12/100
1/1 [==============================] - 0s 48ms/step - loss: 0.5922 - accuracy: 0.7000 - val_loss:
0.5331 - val_accuracy: 0.9800
Epoch 13/100
1/1 [==============================] - 0s 37ms/step - loss: 0.5731 - accuracy: 0.9600 - val_loss:
0.5124 - val_accuracy: 0.7600
Epoch 14/100
1/1 [==============================] - 0s 52ms/step - loss: 0.5568 - accuracy: 0.7000 - val_loss:
0.4993 - val_accuracy: 0.9600
Epoch 15/100
1/1 [==============================] - 0s 36ms/step - loss: 0.5386 - accuracy: 0.9600 - val_loss:
0.4793 - val_accuracy: 0.8000
Epoch 16/100
1/1 [==============================] - 0s 46ms/step - loss: 0.5261 - accuracy: 0.7200 - val_loss:
0.4726 - val_accuracy: 0.9000
Epoch 17/100
1/1 [==============================] - 0s 35ms/step - loss: 0.5126 - accuracy: 0.9500 - val_loss:
0.4527 - val_accuracy: 0.8000
Epoch 18/100
1/1 [==============================] - 0s 36ms/step - loss: 0.5012 - accuracy: 0.7300 - val_loss:
0.4473 - val_accuracy: 0.9000
Epoch 19/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4880 - accuracy: 0.9400 - val_loss:
0.4310 - val_accuracy: 0.8200
Epoch 20/100
1/1 [==============================] - 0s 38ms/step - loss: 0.4802 - accuracy: 0.7600 - val_loss:
0.4278 - val_accuracy: 0.9000
Epoch 21/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4684 - accuracy: 0.9400 - val_loss:
0.4109 - val_accuracy: 0.8400
Epoch 22/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4596 - accuracy: 0.7900 - val_loss:
0.4086 - val_accuracy: 0.9000
Epoch 23/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4500 - accuracy: 0.9500 - val_loss:
0.3924 - val_accuracy: 0.8600
Epoch 24/100
1/1 [==============================] - 0s 34ms/step - loss: 0.4407 - accuracy: 0.8300 - val_loss:
0.3901 - val_accuracy: 0.9000
Epoch 25/100
1/1 [==============================] - 0s 39ms/step - loss: 0.4313 - accuracy: 0.9500 - val_loss:
0.3756 - val_accuracy: 0.8800
Epoch 26/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4233 - accuracy: 0.8500 - val_loss:
0.3737 - val_accuracy: 0.9000
Epoch 27/100
1/1 [==============================] - 0s 35ms/step - loss: 0.4140 - accuracy: 0.9500 - val_loss:
0.3598 - val_accuracy: 0.8800
Epoch 28/100
1/1 [==============================] - 0s 37ms/step - loss: 0.4078 - accuracy: 0.8600 - val_loss:
0.3616 - val_accuracy: 0.9000
Epoch 29/100
1/1 [==============================] - 0s 42ms/step - loss: 0.4011 - accuracy: 0.9500 - val_loss:
0.3477 - val_accuracy: 0.8800
Epoch 30/100
1/1 [==============================] - 0s 37ms/step - loss: 0.3964 - accuracy: 0.8500 - val_loss:
0.3520 - val_accuracy: 0.9000
Epoch 31/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3906 - accuracy: 0.9400 - val_loss:
0.3371 - val_accuracy: 0.8800
Epoch 32/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3862 - accuracy: 0.8500 - val_loss:
0.3414 - val_accuracy: 0.9000
Epoch 33/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3793 - accuracy: 0.9400 - val_loss:
0.3254 - val_accuracy: 0.8800
Epoch 34/100
1/1 [==============================] - 0s 36ms/step - loss: 0.3743 - accuracy: 0.8600 - val_loss:
0.3295 - val_accuracy: 0.9000
Epoch 35/100
1/1 [==============================] - 0s 36ms/step - loss: 0.3670 - accuracy: 0.9400 - val_loss:
0.3133 - val_accuracy: 0.9200
Epoch 36/100
1/1 [==============================] - 0s 47ms/step - loss: 0.3615 - accuracy: 0.8900 - val_loss:
0.3183 - val_accuracy: 0.9000
Epoch 37/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3552 - accuracy: 0.9500 - val_loss:
0.3030 - val_accuracy: 0.9200
Epoch 38/100
1/1 [==============================] - 0s 34ms/step - loss: 0.3505 - accuracy: 0.9100 - val_loss:
0.3074 - val_accuracy: 0.9000
Epoch 39/100
1/1 [==============================] - 0s 39ms/step - loss: 0.3437 - accuracy: 0.9500 - val_loss:
0.2919 - val_accuracy: 0.9400
Epoch 40/100
1/1 [==============================] - 0s 39ms/step - loss: 0.3388 - accuracy: 0.9200 - val_loss:
0.2977 - val_accuracy: 0.9000
Epoch 41/100
1/1 [==============================] - 0s 43ms/step - loss: 0.3329 - accuracy: 0.9500 - val_loss:
0.2832 - val_accuracy: 0.9400
Epoch 42/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3302 - accuracy: 0.9100 - val_loss:
0.2926 - val_accuracy: 0.9000
Epoch 43/100
1/1 [==============================] - 0s 35ms/step - loss: 0.3259 - accuracy: 0.9400 - val_loss:
0.2758 - val_accuracy: 0.9200
Epoch 44/100
1/1 [==============================] - 0s 37ms/step - loss: 0.3236 - accuracy: 0.9100 - val_loss:
0.2861 - val_accuracy: 0.9000
Epoch 45/100
1/1 [==============================] - 0s 34ms/step - loss: 0.3180 - accuracy: 0.9400 - val_loss:
0.2674 - val_accuracy: 0.9200
Epoch 46/100
1/1 [==============================] - 0s 44ms/step - loss: 0.3148 - accuracy: 0.9100 - val_loss:
0.2765 - val_accuracy: 0.9000
Epoch 47/100
1/1 [==============================] - 0s 33ms/step - loss: 0.3078 - accuracy: 0.9500 - val_loss:
0.2575 - val_accuracy: 0.9600
Epoch 48/100
1/1 [==============================] - 0s 34ms/step - loss: 0.3033 - accuracy: 0.9200 - val_loss:
0.2663 - val_accuracy: 0.9000
Epoch 49/100
1/1 [==============================] - 0s 33ms/step - loss: 0.2971 - accuracy: 0.9500 - val_loss:
0.2486 - val_accuracy: 0.9600
Epoch 50/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2934 - accuracy: 0.9200 - val_loss:
0.2586 - val_accuracy: 0.9000
Epoch 51/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2885 - accuracy: 0.9500 - val_loss:
0.2415 - val_accuracy: 0.9600
Epoch 52/100
1/1 [==============================] - 0s 33ms/step - loss: 0.2859 - accuracy: 0.9300 - val_loss:
0.2531 - val_accuracy: 0.9000
Epoch 53/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2818 - accuracy: 0.9500 - val_loss:
0.2356 - val_accuracy: 0.9600
Epoch 54/100
1/1 [==============================] - 0s 39ms/step - loss: 0.2804 - accuracy: 0.9200 - val_loss:
0.2488 - val_accuracy: 0.9000
Epoch 55/100
1/1 [==============================] - 0s 35ms/step - loss: 0.2762 - accuracy: 0.9500 - val_loss:
0.2296 - val_accuracy: 0.9600
Epoch 56/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2746 - accuracy: 0.9200 - val_loss:
0.2435 - val_accuracy: 0.9000
Epoch 57/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2698 - accuracy: 0.9500 - val_loss:
0.2232 - val_accuracy: 0.9600
Epoch 58/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2677 - accuracy: 0.9300 - val_loss:
0.2366 - val_accuracy: 0.9000
Epoch 59/100
1/1 [==============================] - 0s 35ms/step - loss: 0.2622 - accuracy: 0.9500 - val_loss:
0.2162 - val_accuracy: 0.9800
Epoch 60/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2596 - accuracy: 0.9300 - val_loss:
0.2296 - val_accuracy: 0.9000
Epoch 61/100
1/1 [==============================] - 0s 38ms/step - loss: 0.2547 - accuracy: 0.9500 - val_loss:
0.2098 - val_accuracy: 0.9800
Epoch 62/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2525 - accuracy: 0.9300 - val_loss:
0.2240 - val_accuracy: 0.9000
Epoch 63/100
1/1 [==============================] - 0s 40ms/step - loss: 0.2483 - accuracy: 0.9500 - val_loss:
0.2044 - val_accuracy: 0.9800
Epoch 64/100
1/1 [==============================] - 0s 53ms/step - loss: 0.2468 - accuracy: 0.9300 - val_loss:
0.2201 - val_accuracy: 0.9000
Epoch 65/100
1/1 [==============================] - 0s 35ms/step - loss: 0.2431 - accuracy: 0.9500 - val_loss:
0.1996 - val_accuracy: 0.9800
Epoch 66/100
1/1 [==============================] - 0s 41ms/step - loss: 0.2423 - accuracy: 0.9300 - val_loss:
0.2168 - val_accuracy: 0.9000
Epoch 67/100
1/1 [==============================] - 0s 53ms/step - loss: 0.2385 - accuracy: 0.9500 - val_loss:
0.1949 - val_accuracy: 0.9800
Epoch 68/100
1/1 [==============================] - 0s 40ms/step - loss: 0.2375 - accuracy: 0.9300 - val_loss:
0.2125 - val_accuracy: 0.9000
Epoch 69/100
1/1 [==============================] - 0s 37ms/step - loss: 0.2330 - accuracy: 0.9500 - val_loss:
0.1897 - val_accuracy: 0.9800
Epoch 70/100
1/1 [==============================] - 0s 37ms/step - loss: 0.2318 - accuracy: 0.9300 - val_loss:
0.2071 - val_accuracy: 0.9000
Epoch 71/100
1/1 [==============================] - 0s 42ms/step - loss: 0.2267 - accuracy: 0.9500 - val_loss:
0.1838 - val_accuracy: 0.9800
Epoch 72/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2246 - accuracy: 0.9400 - val_loss:
0.2011 - val_accuracy: 0.9000
Epoch 73/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2198 - accuracy: 0.9500 - val_loss:
0.1783 - val_accuracy: 0.9800
Epoch 74/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2185 - accuracy: 0.9400 - val_loss:
0.1971 - val_accuracy: 0.9000
Epoch 75/100
1/1 [==============================] - 0s 34ms/step - loss: 0.2148 - accuracy: 0.9500 - val_loss:
0.1739 - val_accuracy: 0.9800
Epoch 76/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2141 - accuracy: 0.9400 - val_loss:
0.1941 - val_accuracy: 0.9000
Epoch 77/100
1/1 [==============================] - 0s 41ms/step - loss: 0.2107 - accuracy: 0.9500 - val_loss:
0.1701 - val_accuracy: 0.9800
Epoch 78/100
1/1 [==============================] - 0s 35ms/step - loss: 0.2106 - accuracy: 0.9400 - val_loss:
0.1916 - val_accuracy: 0.9000
Epoch 79/100
1/1 [==============================] - 0s 36ms/step - loss: 0.2069 - accuracy: 0.9500 - val_loss:
0.1661 - val_accuracy: 0.9800
Epoch 80/100
1/1 [==============================] - 0s 46ms/step - loss: 0.2066 - accuracy: 0.9400 - val_loss:
0.1883 - val_accuracy: 0.9000
Epoch 81/100
1/1 [==============================] - 0s 38ms/step - loss: 0.2027 - accuracy: 0.9500 - val_loss:
0.1621 - val_accuracy: 0.9800
Epoch 82/100
1/1 [==============================] - 0s 37ms/step - loss: 0.2023 - accuracy: 0.9400 - val_loss:
0.1846 - val_accuracy: 0.9000
Epoch 83/100
1/1 [==============================] - 0s 39ms/step - loss: 0.1982 - accuracy: 0.9500 - val_loss:
0.1581 - val_accuracy: 0.9800
Epoch 84/100
1/1 [==============================] - 0s 36ms/step - loss: 0.1976 - accuracy: 0.9500 - val_loss:
0.1809 - val_accuracy: 0.9000
Epoch 85/100
1/1 [==============================] - 0s 36ms/step - loss: 0.1937 - accuracy: 0.9500 - val_loss:
0.1542 - val_accuracy: 0.9800
Epoch 86/100
1/1 [==============================] - 0s 34ms/step - loss: 0.1932 - accuracy: 0.9500 - val_loss:
0.1776 - val_accuracy: 0.9000
Epoch 87/100
1/1 [==============================] - 0s 36ms/step - loss: 0.1896 - accuracy: 0.9500 - val_loss:
0.1507 - val_accuracy: 0.9800
Epoch 88/100
1/1 [==============================] - 0s 35ms/step - loss: 0.1894 - accuracy: 0.9500 - val_loss:
0.1751 - val_accuracy: 0.9000
Epoch 89/100
1/1 [==============================] - 0s 36ms/step - loss: 0.1861 - accuracy: 0.9500 - val_loss:
0.1474 - val_accuracy: 0.9800
Epoch 90/100
1/1 [==============================] - 0s 39ms/step - loss: 0.1861 - accuracy: 0.9600 - val_loss:
0.1728 - val_accuracy: 0.9000
Epoch 91/100
1/1 [==============================] - 0s 34ms/step - loss: 0.1828 - accuracy: 0.9500 - val_loss:
0.1443 - val_accuracy: 0.9800
Epoch 92/100
1/1 [==============================] - 0s 52ms/step - loss: 0.1830 - accuracy: 0.9600 - val_loss:
0.1707 - val_accuracy: 0.9000
Epoch 93/100
1/1 [==============================] - 0s 37ms/step - loss: 0.1797 - accuracy: 0.9500 - val_loss:
0.1412 - val_accuracy: 0.9800
Epoch 94/100
1/1 [==============================] - 0s 37ms/step - loss: 0.1797 - accuracy: 0.9600 - val_loss:
0.1682 - val_accuracy: 0.9000
Epoch 95/100
1/1 [==============================] - 0s 37ms/step - loss: 0.1764 - accuracy: 0.9600 - val_loss:
0.1382 - val_accuracy: 0.9800
Epoch 96/100
1/1 [==============================] - 0s 35ms/step - loss: 0.1764 - accuracy: 0.9600 - val_loss:
0.1657 - val_accuracy: 0.9000
Epoch 97/100
1/1 [==============================] - 0s 36ms/step - loss: 0.1731 - accuracy: 0.9600 - val_loss:
0.1353 - val_accuracy: 0.9800
Epoch 98/100
1/1 [==============================] - 0s 34ms/step - loss: 0.1731 - accuracy: 0.9600 - val_loss:
0.1632 - val_accuracy: 0.9000
Epoch 99/100
1/1 [==============================] - 0s 56ms/step - loss: 0.1699 - accuracy: 0.9600 - val_loss:
0.1324 - val_accuracy: 0.9800
Epoch 100/100
1/1 [==============================] - 0s 38ms/step - loss: 0.1699 - accuracy: 0.9600 - val_loss:
0.1609 - val_accuracy: 0.9000
```

In [91]:

```
#plot loss
plt.plot(hist.history['loss'], label='loss')
plt.plot(hist.history['val_loss'], label='val_loss')
plt.legend()
```

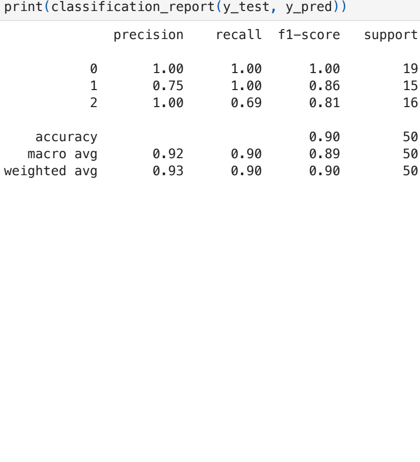
Out[91]: `<matplotlib.legend.Legend at 0x7fd7688fb520>`



In [92]:

```
#Plot accuracy
plt.plot(hist.history['accuracy'],label='acc')
plt.plot(hist.history['val_accuracy'],label='val_acc')
plt.legend()
```

Out[92]: `<matplotlib.legend.Legend at 0x7fd765c4ec40>`



In [95]:

```
# Make predictions
probs = network.predict(X_test)
print(f'Probs shape={probs.shape}')
y_pred = np.argmax(probs,axis=1)
```

```
2/2 [==============================] - 0s 7ms/step
Probs shape=(50, 3)
```

In [97]:

```
# Confussion matrix
show_confusion_matrix(y_test, y_pred, num_classes)
```


In [99]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.75      1.00      0.86        15
           2       1.00      0.69      0.81        16

    accuracy                           0.90        50
   macro avg       0.92      0.90      0.89        50
weighted avg       0.93      0.90      0.90        50
```