DNN_Lab_2024

April 16, 2024

1 Deep Neural Networks Laboration

Data used in this laboration are from the Kitsune Network Attack Dataset, https://archive.ics.uci.edu/ml/datasets/Kitsune+Network+Attack+Dataset . We will focus on the 'Mirai' part of the dataset. Your task is to make a DNN that can classify if each attack is benign or malicious. The dataset has 116 covariates, but to make it a bit more difficult we will remove the first 24 covariates.

You need to answer all questions in this notebook.

If the training is too slow on your own computer, use the smaller datasets (*half or* quarter).

Dense networks are not optimal for tabular datasets like the one used here, but here the main goal is to learn deep learning.

2 Part 1: Get the data

Skip this part if you load stored numpy arrays (Mirai*.npy) (which is recommended)

Use `wget` in the terminal of your cloud machine (in the same directory as where you have saved this notebook) to download the data, i.e.

wget https://archive.ics.uci.edu/ml/machine-learning-databases/00516/mirai/Mirai_dataset.csv.gz

wget https://archive.ics.uci.edu/ml/machine-learning-databases/00516/mirai/Mirai_labels.csv.gz

Then unpack the files using `gunzip` in the terminal, i.e.

gunzip Mirai_dataset.csv.gz

gunzip Mirai_labels.csv.gz

3 Part 2: Get a graphics card

Skip this part if you run on the CPU (recommended)

Lets make sure that our script can see the graphics card that will be used. The graphics cards will perform all the time consuming calculations in every training iteration.

[2]:
```
import os
import warnings
```

```
# Ignore FutureWarning from numpy
warnings.simplefilter(action='ignore', category=FutureWarning)

import keras.backend as K
import tensorflow as tf

os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID";

# The GPU id to use, usually either "0" or "1";
os.environ["CUDA_VISIBLE_DEVICES"]="0";

# Allow growth of GPU memory, otherwise it will always look like all the memory
 ↪is being used
physical_devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[2], line 17
     15 # Allow growth of GPU memory, otherwise it will always look like all the
 ↪memory is being used
     16 physical_devices = tf.config.experimental.list_physical_devices('GPU')
---> 17 tf.config.experimental.set_memory_growth(physical_devices[0], True)

IndexError: list index out of range
```

4 Part 3: Hardware

In deep learning, the computer hardware is very important. You should always know what kind of hardware you are working on. Lets pretend that everyone is using an Nvidia RTX 3090 graphics card.

Question 1: Google the name of the graphics card, how many CUDA cores does it have?

Question 2: How much memory does the graphics card have?

Question 3: What is stored in the GPU memory while training a DNN ?

Answer:

- **Question 1**: 10 496 cores.
- **Question 2**: 24 GB.
- **Question 3**: While training a DNN the GPU memory stores the weights of the model.

5 Part 4: Load the data

To make this step easier, directly load the data from saved numpy arrays (.npy) (recommended)

Load the dataset from the csv files, it will take some time since it is almost 1.4 GB. (not recommended, unless you want to learn how to do it)

We will use the function `genfromtxt` to load the data. (not recommended, unless you want to learn how to do it)

https://docs.scipy.org/doc/numpy/reference/generated/numpy.genfromtxt.html

Load the data from csv files the first time, then save the data as numpy files for faster loading the next time.

Remove the first 24 covariates to make the task harder.

[1]:
```
from numpy import genfromtxt # Not needed if you load data from numpy arrays
import numpy as np
from collections import Counter

# Load data from numpy arrays, choose reduced files if the training takes too␣
↪long
X = np.load('Mirai_data.npy')
Y = np.load('Mirai_labels.npy')



# Remove the first 24 covariates (columns)
X = X[:, 24:]

print('The covariates have size {}.'.format(X.shape))
print('The labels have size {}.'.format(Y.shape))

# Print the number of examples of each class
class_counts = Counter(Y)
for class_label, count in class_counts.items():
    print(f'Class {class_label}: {count} examples')
```

```
The covariates have size (764137, 92).
The labels have size (764137,).
Class 0.0: 121621 examples
Class 1.0: 642516 examples
```

6 Part 5: How good is a naive classifier?

Question 4: Given the number of examples from each class, how high classification performance can a naive classifier obtain? The naive classifier will assume that all examples belong to one class. Note: you do not need to make a naive classifier, this is a theoretical question, just to understand how good performance we can obtain by guessing that all examples belong to one class.

In all classification tasks you should always ask these questions

- How good classification accuracy can a naive classifier obtain? The naive classifier will assume that all examples belong to one class.

- What is random chance classification accuracy if you randomly guess the label of each (test) example? For a balanced dataset and binary classification this is easy (50%), but in many cases it is more complicated and a Monte Carlo simulation may be required to estimate random chance accuracy.

If your classifier cannot perform better than a naive classifier or a random classifier, you are doing something wrong.

[2]:
```python
# It is common to have NaNs in the data, lets check for it. Hint: np.isnan()

# Print the number of NaNs (not a number) in the labels
nan_labels_count = np.isnan(Y).sum()
print(f'The number of NaNs in the labels: {nan_labels_count}')

# Print the number of NaNs in the covariates
nan_covariates_count = np.isnan(X).sum()
print(f'The number of NaNs in the covariates: {nan_covariates_count}')
```

```
The number of NaNs in the labels: 0
The number of NaNs in the covariates: 0
```

Answer: Question 4

- A Naive classification would in our case guess class 1. Which will give a classification accuracy 642516 / (642516 + 121621) = 0.84
- With random chance we should be able to correctly classify 50% of each class for binary classes. If we have n classes we would be able to classify 1/n of each class.

7 Part 6: Preprocessing

Lets do some simple preprocessing

[3]:
```python
# Convert covariates to floats
X = X.astype(float)

# Convert labels to integers
Y = Y.astype(int)

# Remove mean of each covariate (column)
X_mean = np.mean(X, axis=0)
X -= X_mean.reshape(1, -1)

# Divide each covariate (column) by its standard deviation
X_std = np.std(X, axis=0)
X /= X_std.reshape(1, -1)
```

```python
# Check that mean is 0 and standard deviation is 1 for all covariates, by␣
↪printing mean and std
print("Mean of each covariate after preprocessing:")
print(np.mean(X, axis=0))
print("\nStandard deviation of each covariate after preprocessing:")
print(np.std(X, axis=0))
```

```
Mean of each covariate after preprocessing:
[-3.19451533e-18 -6.32970181e-14  1.19926356e-13  4.56743018e-15
  4.10210037e-14  1.46130975e-13  5.85246484e-16 -1.69734859e-14
 -3.36915700e-13  1.28688437e-12 -2.69360995e-12 -1.10733213e-13
 -1.22392702e-13 -1.70649630e-13 -1.02461166e-14  2.50701280e-12
  1.47553162e-12  1.08446837e-12 -1.04981959e-13  6.83458762e-14
 -1.03373555e-13  5.98825773e-14 -1.02025960e-12 -1.68983055e-12
 -1.79101143e-12 -1.31828514e-13  4.42580403e-13  6.14635580e-13
  5.78048199e-14 -4.92623328e-13 -2.54513072e-12  1.86544900e-13
 -1.53444593e-13  1.68079591e-12  9.30041709e-13  1.50738177e-13
 -1.15688852e-12 -3.62610361e-13 -1.71390937e-12 -2.09264067e-13
  1.07161976e-12 -1.45236885e-12 -1.69724579e-14 -1.64918984e-16
 -5.13444996e-14 -1.02171349e-14 -1.74685907e-15  1.34264921e-13
  5.98801969e-14  1.48745574e-17 -4.25442340e-13  5.78079594e-14
  1.25638129e-15  1.69449684e-13  1.50725881e-13  2.14439542e-14
  3.65457183e-14  1.17260451e-13 -8.82752870e-13 -6.34816648e-13
 -1.62109649e-12  2.63270303e-13 -7.57215123e-15 -2.89395002e-14
 -3.90180996e-13 -1.53167085e-12 -9.57913621e-13  2.47411065e-13
  2.44200541e-13 -6.73050928e-15  1.07502596e-13  2.58222203e-13
 -1.87714601e-13 -1.19882476e-12 -2.17154862e-12  5.48444735e-14
  5.46183481e-15  3.71315442e-14  1.47576646e-13 -1.62639245e-12
 -1.23986972e-13 -1.71744315e-12  5.29956657e-13 -3.21442452e-14
 -4.59767392e-14  3.56347870e-13 -1.48544246e-12 -1.26642728e-13
  1.52633871e-13  9.58048710e-14  4.34603426e-14 -4.07615740e-14]

Standard deviation of each covariate after preprocessing:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

8 Part 7: Split the dataset

Use the first 70% of the dataset for training, leave the other 30% for validation and test, call the variables

Xtrain (70%)

Xtemp (30%)

Ytrain (70%)

Ytemp (30%)

We use a function from scikit learn. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.tra

[4]:
```
from sklearn.model_selection import train_test_split

# Your code to split the dataset
Xtrain, Xtemp, Ytrain, Ytemp = train_test_split(X, Y, test_size=0.3,␣
 ↪random_state=13)




print('Xtrain has size {}.'.format(Xtrain.shape))
print('Ytrain has size {}.'.format(Ytrain.shape))

print('Xtemp has size {}.'.format(Xtemp.shape))
print('Ytemp has size {}.'.format(Ytemp.shape))

# Print the number of examples of each class, for the training data and the␣
 ↪remaining 30%
# Training
train_class_counts = Counter(Ytrain)
print("\nNumber of examples of each class in the training data:")
for class_label, count in train_class_counts.items():
    print(f'Class {class_label}: {count} examples')


# Test & Validation
temp_class_counts = Counter(Ytemp)
print("\nNumber of examples of each class in the remaining 30% data:")
for class_label, count in temp_class_counts.items():
    print(f'Class {class_label}: {count} examples')
```

```
Xtrain has size (534895, 92).
Ytrain has size (534895,).
Xtemp has size (229242, 92).
Ytemp has size (229242,).

Number of examples of each class in the training data:
Class 1: 449824 examples
Class 0: 85071 examples

Number of examples of each class in the remaining 30% data:
Class 1: 192692 examples
Class 0: 36550 examples
```

9 Part 8: Split non-training data data into validation and test

Now split your non-training data (Xtemp, Ytemp) into 50% validation (Xval, Yval) and 50% testing (Xtest, Ytest), we use a function from scikit learn. In total this gives us 70% for training, 15% for validation, 15% for test.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Do all variables (Xtrain,Ytrain), (Xval,Yval), (Xtest,Ytest) have the shape that you expect?

[5]:
```
from sklearn.model_selection import train_test_split

# Your code
Xval, Xtest, Yval, Ytest = train_test_split(Xtemp, Ytemp, test_size=0.5,␣
↪random_state=13)

print('We get: \nX-validation:{}, \nY-validation:{}, \nX-test:{}, \nY-test:{}'.
↪format(Xval.shape, Yval.shape, Xtest.shape, Ytest.shape))
```

```
We get:
X-validation:(114621, 92),
Y-validation:(114621,),
X-test:(114621, 92),
Y-test:(114621,)
```

Answer: It is expected that training-, validation-, and test data have 92 features. We also expect that 70% of the total number of observations is in train data and 15% for validation and 15% for test data.

10 Part 9: DNN classification

Finish this code to create a first version of the classifier using a DNN. Start with a simple network with 2 dense layers (with 20 nodes each), using sigmoid activation functions. The final dense layer should have a single node and a sigmoid activation function. We start with the SGD optimizer.

For different parts of this notebook you need to go back here, add more things, and re-run this cell to re-define the build function.

Relevant functions are

`model.add()`, adds a layer to the network

`Dense()`, a dense network layer

`model.compile()`, compile the model, add " metrics=[‘accuracy’] " to print the classification accuracy during the training

See https://keras.io/layers/core/ for information on how the `Dense()` function works

Import a relevant cost / loss function for binary classification from keras.losses (https://keras.io/losses/)

See the following links for how to compile, train and evaluate the model https://keras.io/api/models/model_training_apis/#compile-method

https://keras.io/api/models/model_training_apis/#fit-method

https://keras.io/api/models/model_training_apis/#evaluate-method

Make sure that the last layer always has a sigmoid activation function (why?).

[6]:
```
from keras.models import Sequential, Model
from keras.layers import Input, Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from keras.losses import BinaryCrossentropy as BC


# Set seed from random number generator, for better comparisons
from numpy.random import seed
seed(123)

def build_DNN(input_shape, n_layers, n_nodes, act_fun='sigmoid',␣
 ↪optimizer='sgd', learning_rate=0.01,
              batch_norm = False, dropout = False, mydropout = False):

    # Setup optimizer, depending on input parameter string
    if optimizer == 'sgd':
        optimizer = SGD(learning_rate=learning_rate)
    elif optimizer == 'Adam':
        optimizer = Adam(learning_rate=learning_rate)
    else:
        raise ValueError("Optimizer not recognized")

    # Setup a sequential model
    model = Sequential()

    # Add layers to the model, using the input parameters of the build_DNN␣
 ↪function

    # Add first layer, requires input shape
    model.add(Dense(n_nodes, activation=act_fun, input_dim=input_shape))
    if(batch_norm == True):
        model.add(BatchNormalization())
    if(dropout == True):
        model.add(Dropout(0.5))
    if(mydropout == True):
        model.add(myDropout(0.5))


    # Add remaining layers, do not require input shape
    for i in range(n_layers-1):
        model.add(Dense(n_nodes, activation=act_fun))
```

```
        if(batch_norm == True):
            model.add(BatchNormalization())
        if(dropout == True):
            model.add(Dropout(0.5))
        if(mydropout == True):
            model.add(myDropout(0.5))

    # Add final layer
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
    model.compile(loss=BC(), optimizer=SGD(learning_rate=learning_rate),␣
 ↪metrics=['accuracy'])

    return model
```

[7]:
```
# Lets define a help function for plotting the training results

import matplotlib.pyplot as plt
def plot_results(history):

    val_loss = history.history['val_loss']
    acc = history.history['accuracy']
    loss = history.history['loss']
    val_acc = history.history['val_accuracy']

    plt.figure(figsize=(10,4))
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(loss)
    plt.plot(val_loss)
    plt.legend(['Training','Validation'])

    plt.figure(figsize=(10,4))
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(acc)
    plt.plot(val_acc)
    plt.legend(['Training','Validation'])

    plt.show()
```

11 Part 10: Train the DNN

Time to train the DNN, we start simple with 2 layers with 20 nodes each, learning rate 0.1.

Relevant functions

`build_DNN`, the function we defined in Part 9, call it with the parameters you want to use

`model.fit()`, train the model with some training data

`model.evaluate()`, apply the trained model to some test data

See the following links for how to train and evaluate the model

https://keras.io/api/models/model_training_apis/#fit-method

https://keras.io/api/models/model_training_apis/#evaluate-method

Make sure that you are using learning rate 0.1 !

11.0.1 2 layers, 20 nodes

[8]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20

# Number of features = 92
input_shape = 92
n_layers = 2
n_nodes = 20

# Build the model
model1 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes=20)

# Train the model, provide training data and validation data
history1 = model1.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),
 ↪batch_size = batch_size, epochs = epochs)
```

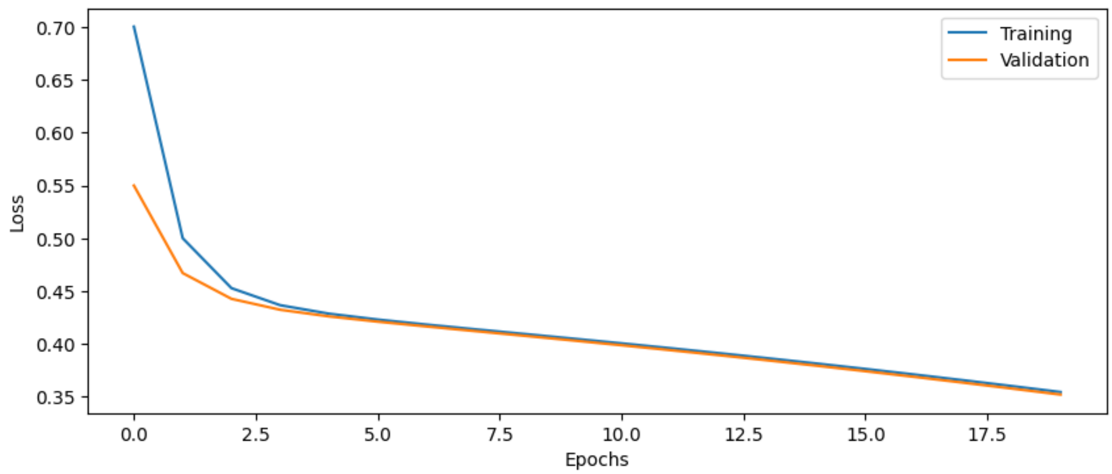
```
Epoch 1/20
54/54 [==============================] - 1s 9ms/step - loss: 0.7005 - accuracy:
0.5339 - val_loss: 0.5499 - val_accuracy: 0.8406
Epoch 2/20
54/54 [==============================] - 0s 6ms/step - loss: 0.5001 - accuracy:
0.8410 - val_loss: 0.4670 - val_accuracy: 0.8406
Epoch 3/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4527 - accuracy:
0.8410 - val_loss: 0.4426 - val_accuracy: 0.8406
Epoch 4/20
54/54 [==============================] - 0s 5ms/step - loss: 0.4366 - accuracy:
0.8410 - val_loss: 0.4323 - val_accuracy: 0.8406
Epoch 5/20
54/54 [==============================] - 0s 5ms/step - loss: 0.4285 - accuracy:
0.8410 - val_loss: 0.4260 - val_accuracy: 0.8406
Epoch 6/20
54/54 [==============================] - 0s 5ms/step - loss: 0.4230 - accuracy:
0.8410 - val_loss: 0.4210 - val_accuracy: 0.8406
Epoch 7/20
```

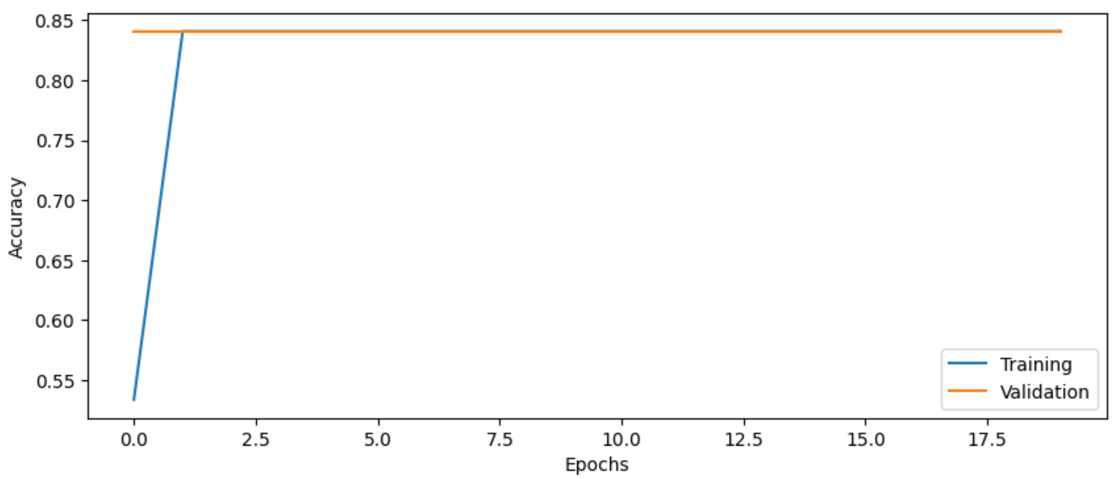
```
54/54 [==============================] - 0s 7ms/step - loss: 0.4182 - accuracy:
0.8410 - val_loss: 0.4164 - val_accuracy: 0.8406
Epoch 8/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4138 - accuracy:
0.8410 - val_loss: 0.4120 - val_accuracy: 0.8406
Epoch 9/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4094 - accuracy:
0.8410 - val_loss: 0.4076 - val_accuracy: 0.8406
Epoch 10/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4049 - accuracy:
0.8410 - val_loss: 0.4032 - val_accuracy: 0.8406
Epoch 11/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4004 - accuracy:
0.8410 - val_loss: 0.3986 - val_accuracy: 0.8406
Epoch 12/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3958 - accuracy:
0.8410 - val_loss: 0.3940 - val_accuracy: 0.8406
Epoch 13/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3911 - accuracy:
0.8410 - val_loss: 0.3892 - val_accuracy: 0.8406
Epoch 14/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3863 - accuracy:
0.8410 - val_loss: 0.3843 - val_accuracy: 0.8406
Epoch 15/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3813 - accuracy:
0.8410 - val_loss: 0.3792 - val_accuracy: 0.8406
Epoch 16/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3762 - accuracy:
0.8410 - val_loss: 0.3741 - val_accuracy: 0.8406
Epoch 17/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3709 - accuracy:
0.8410 - val_loss: 0.3687 - val_accuracy: 0.8406
Epoch 18/20
54/54 [==============================] - 0s 5ms/step - loss: 0.3655 - accuracy:
0.8410 - val_loss: 0.3632 - val_accuracy: 0.8406
Epoch 19/20
54/54 [==============================] - 0s 6ms/step - loss: 0.3600 - accuracy:
0.8410 - val_loss: 0.3576 - val_accuracy: 0.8406
Epoch 20/20
54/54 [==============================] - 0s 5ms/step - loss: 0.3543 - accuracy:
0.8410 - val_loss: 0.3519 - val_accuracy: 0.8406
```

[9]:
```
# Evaluate the model on the test data
score = model1.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 3s 968us/step - loss: 0.3523 -
accuracy: 0.8405
Test loss: 0.3523
Test accuracy: 0.8405
```

[10]:
```
# Plot the history from the training run
plot_results(history1)
```





12 Part 11: More questions

Question 5: What happens if you add several Dense layers without specifying the activation function?

Question 6: How are the weights in each dense layer initialized as default? How are the bias weights initialized?

Answer: **Question 5:** When we add several Dense layers without specifying the activation function, keras will use no activation function. Mathematically we could remove these layers and add more nodes to the last layer that had an activation function.

Question 6: The weights are initialized with glorot uniform by default. The bias weights are initialized as zeros.

13 Part 12: Balancing the classes

This dataset is rather unbalanced, we need to define class weights so that the training pays more attention to the class with fewer samples. We use a function in scikit learn

https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html

You need to call the function something like this

class_weights = class_weight.compute_class_weight(class_weight = , classes = , y =)

otherwise it will complain

[11]:
```
from sklearn.utils import class_weight

# Calculate class weights
weights = class_weight.compute_class_weight(class_weight = "balanced", classes␣
↪= np.unique(Ytrain), y = Ytrain)

# Print the class weights
print(f"Weights for class 0: {round(weights[0],4)}")
print(f"Weights for class 1: {round(weights[1],4)}")
# Keras wants the weights in this form, uncomment and change value1 and value2␣
↪to your weights,
# or get them from the array that is returned from class_weight

class_weights = {0: weights[0],
                 1: weights[1]}
print(class_weights)
```

```
Weights for class 0: 3.1438
Weights for class 1: 0.5946
{0: 3.143815166155329, 1: 0.5945603169239525}
```

13.0.1 2 layers, 20 nodes, class weights

[12]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92
```

```
# Build and train model
model2 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes=20)

history2 = model2.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),
    class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 1s 9ms/step - loss: 0.6982 - accuracy:
0.8447 - val_loss: 0.6842 - val_accuracy: 0.8523
Epoch 2/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6882 - accuracy:
0.8540 - val_loss: 0.6825 - val_accuracy: 0.8588
Epoch 3/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6787 - accuracy:
0.8683 - val_loss: 0.6768 - val_accuracy: 0.8767
Epoch 4/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6693 - accuracy:
0.8801 - val_loss: 0.6691 - val_accuracy: 0.8802
Epoch 5/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6598 - accuracy:
0.8824 - val_loss: 0.6591 - val_accuracy: 0.8813
Epoch 6/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6499 - accuracy:
0.8828 - val_loss: 0.6499 - val_accuracy: 0.8828
Epoch 7/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6396 - accuracy:
0.8831 - val_loss: 0.6396 - val_accuracy: 0.8817
Epoch 8/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6287 - accuracy:
0.8825 - val_loss: 0.6287 - val_accuracy: 0.8808
Epoch 9/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6172 - accuracy:
0.8820 - val_loss: 0.6173 - val_accuracy: 0.8803
Epoch 10/20
54/54 [==============================] - 0s 6ms/step - loss: 0.6050 - accuracy:
0.8812 - val_loss: 0.6054 - val_accuracy: 0.8790
Epoch 11/20
54/54 [==============================] - 0s 6ms/step - loss: 0.5920 - accuracy:
0.8800 - val_loss: 0.5921 - val_accuracy: 0.8783
Epoch 12/20
54/54 [==============================] - 0s 7ms/step - loss: 0.5782 - accuracy:
0.8793 - val_loss: 0.5784 - val_accuracy: 0.8778
Epoch 13/20
54/54 [==============================] - 0s 6ms/step - loss: 0.5636 - accuracy:
0.8792 - val_loss: 0.5641 - val_accuracy: 0.8780
Epoch 14/20
54/54 [==============================] - 0s 6ms/step - loss: 0.5483 - accuracy:
```

```
0.8792 - val_loss: 0.5490 - val_accuracy: 0.8779
Epoch 15/20
54/54 [==============================] - 0s 6ms/step - loss: 0.5322 - accuracy:
0.8792 - val_loss: 0.5334 - val_accuracy: 0.8780
Epoch 16/20
54/54 [==============================] - 0s 6ms/step - loss: 0.5156 - accuracy:
0.8792 - val_loss: 0.5175 - val_accuracy: 0.8781
Epoch 17/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4985 - accuracy:
0.8793 - val_loss: 0.5012 - val_accuracy: 0.8782
Epoch 18/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4812 - accuracy:
0.8794 - val_loss: 0.4849 - val_accuracy: 0.8783
Epoch 19/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4638 - accuracy:
0.8795 - val_loss: 0.4686 - val_accuracy: 0.8784
Epoch 20/20
54/54 [==============================] - 1s 20ms/step - loss: 0.4465 - accuracy:
0.8798 - val_loss: 0.4526 - val_accuracy: 0.8788
```

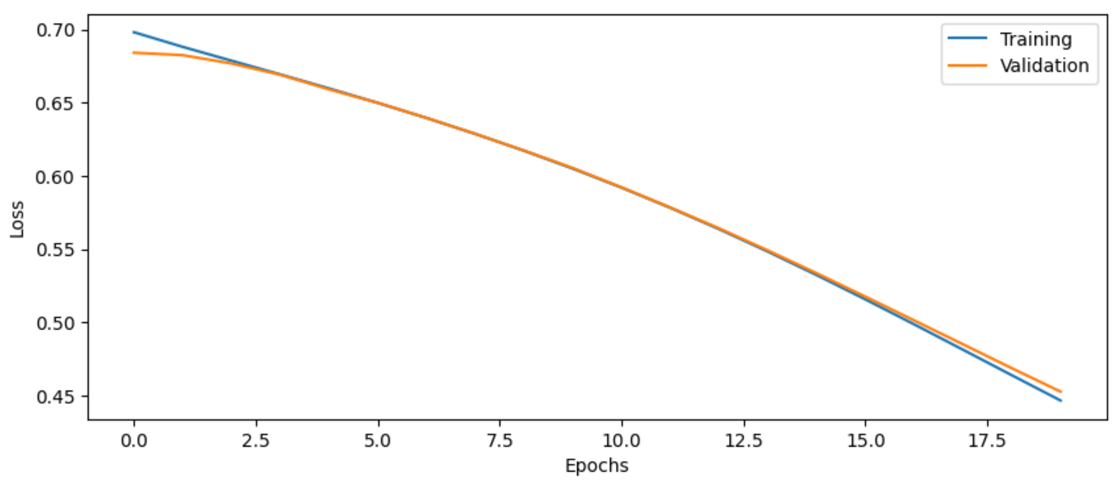
[13]:

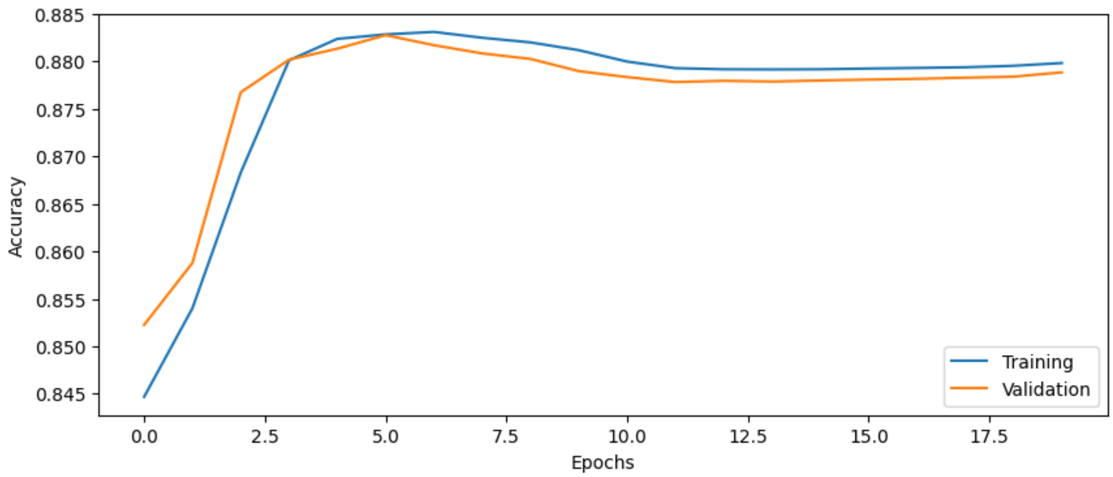
```
# Evaluate model on test data
score = model2.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 4s 975us/step - loss: 0.4519 -
accuracy: 0.8799
Test loss: 0.4519
Test accuracy: 0.8799
```

[14]:

```
plot_results(history2)
```





14 Part 13: More questions

Skip questions 8 and 9 if you run on the CPU (recommended)

Question 7: Why do we have to use a batch size? Why can't we simply use all data at once? This is more relevant for even larger datasets.

Question 8: How busy is the GPU for a batch size of 100? How much GPU memory is used? Hint: run 'nvidia-smi' on the computer a few times during training.

Question 9: What is the processing time for one training epoch when the batch size is 100? What is the processing time for one epoch when the batch size is 1,000? What is the processing time for one epoch when the batch size is 10,000? Explain the results.

Question 10: How many times are the weights in the DNN updated in each training epoch if the batch size is 100? How many times are the weights in the DNN updated in each training epoch if the batch size is 1,000? How many times are the weights in the DNN updated in each training epoch if the batch size is 10,000?

Question 11: What limits how large the batch size can be?

Question 12: Generally speaking, how is the learning rate related to the batch size? If the batch size is decreased, how should the learning rate be changed?

Lets use a batch size of 10,000 from now on, and a learning rate of 0.1.

Answer:

- **Question 7**: In case we have a large data set, all of the data might not be able to fit in CPU, so without batch size the training would crash. With batch size, we update the weights after the model has seen batch size number of observations, which can speed up training.
- **Question 10**: The number of observations in training data is 534895.

  - With batch size of 100, we will update the weights 534895/100 = 5349 times per epoch.
  - With batch size of 1000, we will update the weights 534895/1000 = 535 times per epoch..
  - With batch size of 10000, we will update the weights 534895/10000 = 54 times per epoch.

- **Question 11**: We run on CPU, so we are limited by the memory of the CPU. So the size of the data with batch size can not be larger than the memory.
- **Question 12**: With low batch size, we should have small learning rate since the gradient is more uncertain. With large batch size we should increase learning rate, otherwise training will take longer.

15 Part 14: Increasing the complexity

Lets try some different configurations of number of layers and number of nodes per layer.

Question 13: How many trainable parameters does the network with 4 dense layers with 50 nodes each have, compared to the initial network with 2 layers and 20 nodes per layer? Hint: use model.summary()

15.0.1 4 layers, 20 nodes, class weights

[15]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model3 = build_DNN(input_shape = input_shape, n_layers = 4, n_nodes = 20,␣
 ↪learning_rate = 0.1)

history3 = model3.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
 ↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 1s 11ms/step - loss: 0.6989 - accuracy:
0.7474 - val_loss: 0.6961 - val_accuracy: 0.1594
Epoch 2/20
54/54 [==============================] - 1s 18ms/step - loss: 0.6922 - accuracy:
0.5563 - val_loss: 0.6958 - val_accuracy: 0.1594
Epoch 3/20
54/54 [==============================] - 0s 8ms/step - loss: 0.6918 - accuracy:
0.5703 - val_loss: 0.6950 - val_accuracy: 0.1598
Epoch 4/20
54/54 [==============================] - 0s 8ms/step - loss: 0.6915 - accuracy:
0.5260 - val_loss: 0.6873 - val_accuracy: 0.8406
Epoch 5/20
54/54 [==============================] - 0s 8ms/step - loss: 0.6910 - accuracy:
```

```
0.8147 - val_loss: 0.6934 - val_accuracy: 0.1635
Epoch 6/20
54/54 [==============================] - 0s 8ms/step - loss: 0.6905 - accuracy:
0.7396 - val_loss: 0.6918 - val_accuracy: 0.8421
Epoch 7/20
54/54 [==============================] - 0s 8ms/step - loss: 0.6898 - accuracy:
0.7889 - val_loss: 0.6855 - val_accuracy: 0.8550
Epoch 8/20
54/54 [==============================] - 1s 10ms/step - loss: 0.6890 - accuracy:
0.8783 - val_loss: 0.6869 - val_accuracy: 0.8949
Epoch 9/20
54/54 [==============================] - 0s 9ms/step - loss: 0.6879 - accuracy:
0.8784 - val_loss: 0.6849 - val_accuracy: 0.8936
Epoch 10/20
54/54 [==============================] - 1s 14ms/step - loss: 0.6866 - accuracy:
0.8756 - val_loss: 0.6834 - val_accuracy: 0.8990
Epoch 11/20
54/54 [==============================] - 1s 10ms/step - loss: 0.6848 - accuracy:
0.8827 - val_loss: 0.6798 - val_accuracy: 0.8957
Epoch 12/20
54/54 [==============================] - 0s 9ms/step - loss: 0.6824 - accuracy:
0.8907 - val_loss: 0.6743 - val_accuracy: 0.8835
Epoch 13/20
54/54 [==============================] - 0s 9ms/step - loss: 0.6791 - accuracy:
0.8962 - val_loss: 0.6766 - val_accuracy: 0.8957
Epoch 14/20
54/54 [==============================] - 0s 8ms/step - loss: 0.6742 - accuracy:
0.8924 - val_loss: 0.6647 - val_accuracy: 0.8997
Epoch 15/20
54/54 [==============================] - 1s 12ms/step - loss: 0.6670 - accuracy:
0.8955 - val_loss: 0.6614 - val_accuracy: 0.8948
Epoch 16/20
54/54 [==============================] - 0s 7ms/step - loss: 0.6557 - accuracy:
0.8942 - val_loss: 0.6440 - val_accuracy: 0.8962
Epoch 17/20
54/54 [==============================] - 1s 9ms/step - loss: 0.6369 - accuracy:
0.8932 - val_loss: 0.6264 - val_accuracy: 0.8863
Epoch 18/20
54/54 [==============================] - 1s 11ms/step - loss: 0.6044 - accuracy:
0.8897 - val_loss: 0.5832 - val_accuracy: 0.8853
Epoch 19/20
54/54 [==============================] - 1s 14ms/step - loss: 0.5475 - accuracy:
0.8859 - val_loss: 0.5022 - val_accuracy: 0.8863
Epoch 20/20
54/54 [==============================] - 1s 14ms/step - loss: 0.4581 - accuracy:
0.8836 - val_loss: 0.4119 - val_accuracy: 0.8813
```

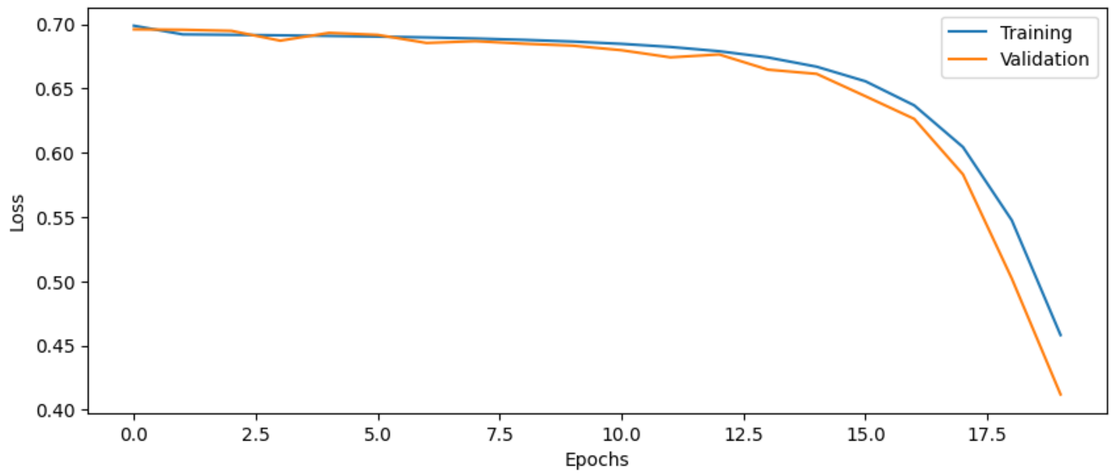
[16]:

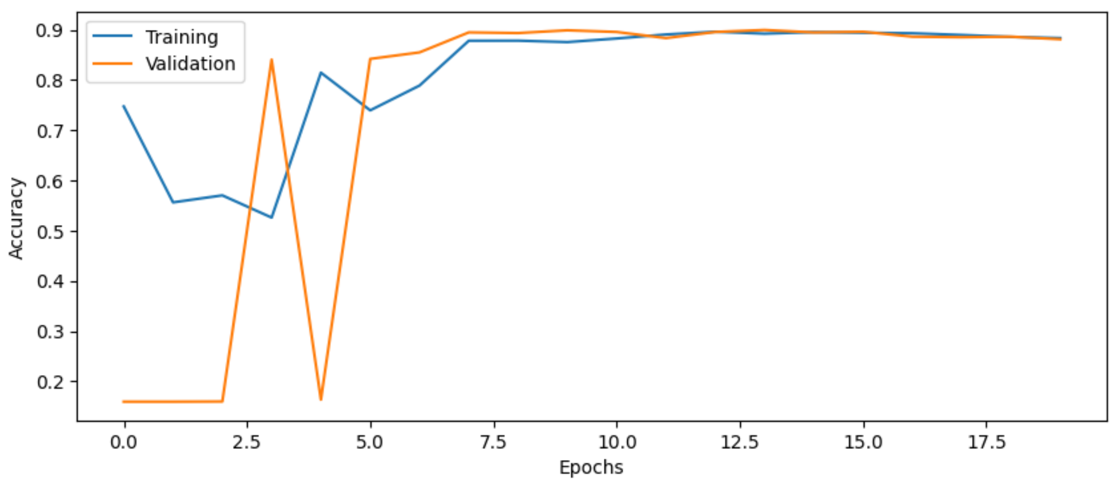
```
# Evaluate model on test data
score = model3.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 4s 1ms/step - loss: 0.4111 -
accuracy: 0.8820
Test loss: 0.4111
Test accuracy: 0.8820
```

[17]:

```
plot_results(history3)
```





15.0.2 2 layers, 50 nodes, class weights

[18]:
```python
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model4 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes = 50,␣
 ↪learning_rate = 0.1)

history4 = model4.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
 ↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 1s 13ms/step - loss: 0.5955 - accuracy:
0.8814 - val_loss: 0.4975 - val_accuracy: 0.8822
Epoch 2/20
54/54 [==============================] - 1s 10ms/step - loss: 0.3850 - accuracy:
0.8829 - val_loss: 0.3294 - val_accuracy: 0.8818
Epoch 3/20
54/54 [==============================] - 1s 14ms/step - loss: 0.2694 - accuracy:
0.8831 - val_loss: 0.2857 - val_accuracy: 0.8835
Epoch 4/20
54/54 [==============================] - 1s 10ms/step - loss: 0.2341 - accuracy:
0.8859 - val_loss: 0.2730 - val_accuracy: 0.8866
Epoch 5/20
54/54 [==============================] - 1s 15ms/step - loss: 0.2208 - accuracy:
0.8892 - val_loss: 0.2638 - val_accuracy: 0.8904
Epoch 6/20
54/54 [==============================] - 1s 20ms/step - loss: 0.2136 - accuracy:
0.8919 - val_loss: 0.2605 - val_accuracy: 0.8917
Epoch 7/20
54/54 [==============================] - 1s 15ms/step - loss: 0.2087 - accuracy:
0.8935 - val_loss: 0.2550 - val_accuracy: 0.8942
Epoch 8/20
54/54 [==============================] - 1s 13ms/step - loss: 0.2050 - accuracy:
0.8955 - val_loss: 0.2559 - val_accuracy: 0.8950
Epoch 9/20
54/54 [==============================] - 1s 12ms/step - loss: 0.2020 - accuracy:
0.8964 - val_loss: 0.2512 - val_accuracy: 0.8959
Epoch 10/20
54/54 [==============================] - 1s 17ms/step - loss: 0.1994 - accuracy:
0.8970 - val_loss: 0.2494 - val_accuracy: 0.8962
Epoch 11/20
54/54 [==============================] - 1s 18ms/step - loss: 0.1971 - accuracy:
0.8973 - val_loss: 0.2465 - val_accuracy: 0.8966
Epoch 12/20
```

```
54/54 [==============================] - 1s 20ms/step - loss: 0.1950 - accuracy:
0.8976 - val_loss: 0.2449 - val_accuracy: 0.8968
Epoch 13/20
54/54 [==============================] - 1s 24ms/step - loss: 0.1933 - accuracy:
0.8978 - val_loss: 0.2420 - val_accuracy: 0.8971
Epoch 14/20
54/54 [==============================] - 1s 21ms/step - loss: 0.1917 - accuracy:
0.8982 - val_loss: 0.2403 - val_accuracy: 0.8976
Epoch 15/20
54/54 [==============================] - 1s 12ms/step - loss: 0.1904 - accuracy:
0.8989 - val_loss: 0.2389 - val_accuracy: 0.8984
Epoch 16/20
54/54 [==============================] - 1s 12ms/step - loss: 0.1892 - accuracy:
0.8997 - val_loss: 0.2377 - val_accuracy: 0.8992
Epoch 17/20
54/54 [==============================] - 1s 13ms/step - loss: 0.1881 - accuracy:
0.9006 - val_loss: 0.2381 - val_accuracy: 0.8998
Epoch 18/20
54/54 [==============================] - 1s 14ms/step - loss: 0.1871 - accuracy:
0.9012 - val_loss: 0.2373 - val_accuracy: 0.9003
Epoch 19/20
54/54 [==============================] - 1s 19ms/step - loss: 0.1862 - accuracy:
0.9019 - val_loss: 0.2339 - val_accuracy: 0.9015
Epoch 20/20
54/54 [==============================] - 1s 14ms/step - loss: 0.1854 - accuracy:
0.9026 - val_loss: 0.2348 - val_accuracy: 0.9018
```

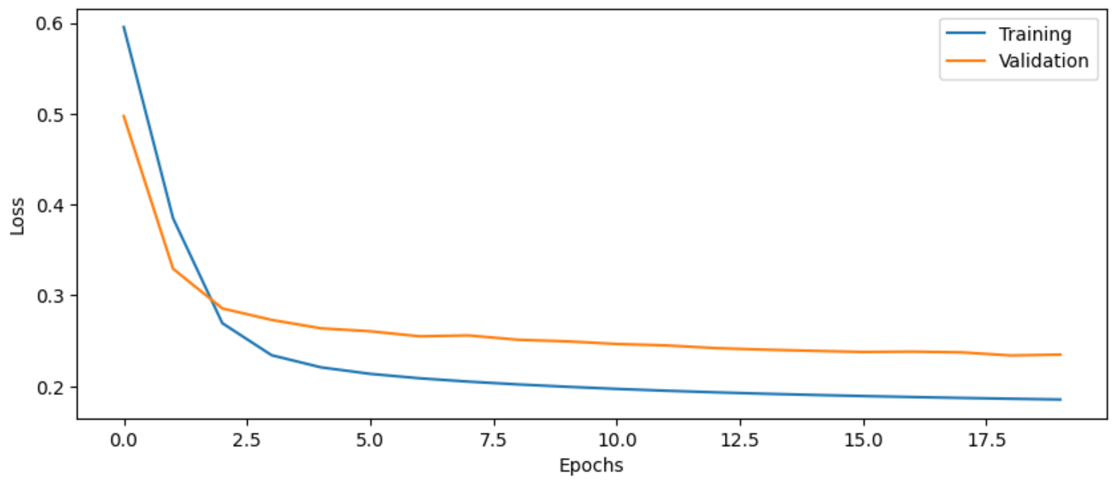
[19]:

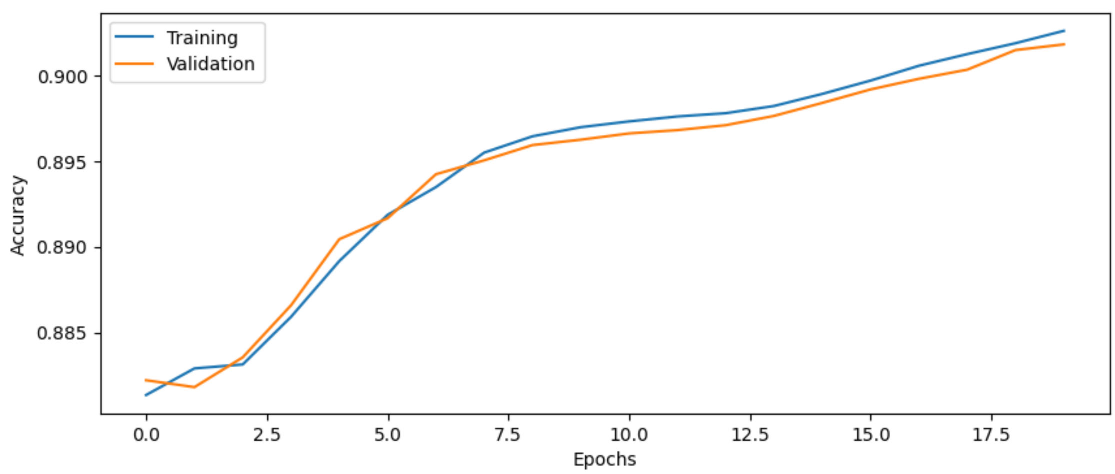
```python
# Evaluate model on test data
score = model4.evaluate(Xtest, Ytest)


print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2314 -
accuracy: 0.9034
Test loss: 0.2314
Test accuracy: 0.9034
```

[20]:

```python
plot_results(history4)
```





15.0.3 4 layers, 50 nodes, class weights

[21]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model5 = build_DNN(input_shape = input_shape, n_layers = 4, n_nodes = 50,␣
↪learning_rate = 0.1)

history5 = model5.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 3s 36ms/step - loss: 0.6944 - accuracy:
0.4380 - val_loss: 0.7014 - val_accuracy: 0.1594
Epoch 2/20
54/54 [==============================] - 11s 210ms/step - loss: 0.6925 -
accuracy: 0.5113 - val_loss: 0.6872 - val_accuracy: 0.8406
Epoch 3/20
54/54 [==============================] - 2s 40ms/step - loss: 0.6918 - accuracy:
0.5651 - val_loss: 0.6859 - val_accuracy: 0.8406
Epoch 4/20
54/54 [==============================] - 1s 27ms/step - loss: 0.6910 - accuracy:
0.6956 - val_loss: 0.6924 - val_accuracy: 0.8188
Epoch 5/20
54/54 [==============================] - 1s 26ms/step - loss: 0.6902 - accuracy:
0.6705 - val_loss: 0.6813 - val_accuracy: 0.8406
Epoch 6/20
54/54 [==============================] - 13s 240ms/step - loss: 0.6891 -
accuracy: 0.7700 - val_loss: 0.6926 - val_accuracy: 0.5614
Epoch 7/20
54/54 [==============================] - 11s 210ms/step - loss: 0.6877 -
accuracy: 0.7290 - val_loss: 0.6848 - val_accuracy: 0.8894
Epoch 8/20
54/54 [==============================] - 13s 249ms/step - loss: 0.6860 -
accuracy: 0.8232 - val_loss: 0.6851 - val_accuracy: 0.8901
Epoch 9/20
54/54 [==============================] - 1s 21ms/step - loss: 0.6836 - accuracy:
0.8681 - val_loss: 0.6903 - val_accuracy: 0.7881
Epoch 10/20
54/54 [==============================] - 13s 237ms/step - loss: 0.6802 -
accuracy: 0.8839 - val_loss: 0.6702 - val_accuracy: 0.8877
Epoch 11/20
54/54 [==============================] - 1s 25ms/step - loss: 0.6751 - accuracy:
0.8877 - val_loss: 0.6652 - val_accuracy: 0.8977
Epoch 12/20
54/54 [==============================] - 2s 33ms/step - loss: 0.6671 - accuracy:
0.8900 - val_loss: 0.6591 - val_accuracy: 0.8919
Epoch 13/20
54/54 [==============================] - 2s 32ms/step - loss: 0.6534 - accuracy:
0.8884 - val_loss: 0.6498 - val_accuracy: 0.8811
Epoch 14/20
54/54 [==============================] - 2s 29ms/step - loss: 0.6279 - accuracy:
0.8861 - val_loss: 0.6115 - val_accuracy: 0.8827
Epoch 15/20
54/54 [==============================] - 2s 35ms/step - loss: 0.5766 - accuracy:
0.8845 - val_loss: 0.5475 - val_accuracy: 0.8811
Epoch 16/20
54/54 [==============================] - 2s 33ms/step - loss: 0.4782 - accuracy:
0.8827 - val_loss: 0.4259 - val_accuracy: 0.8809
```

```
Epoch 17/20
54/54 [==============================] - 2s 29ms/step - loss: 0.3501 - accuracy:
0.8821 - val_loss: 0.3236 - val_accuracy: 0.8811
Epoch 18/20
54/54 [==============================] - 1s 22ms/step - loss: 0.2667 - accuracy:
0.8825 - val_loss: 0.2871 - val_accuracy: 0.8818
Epoch 19/20
54/54 [==============================] - 1s 25ms/step - loss: 0.2332 - accuracy:
0.8844 - val_loss: 0.2706 - val_accuracy: 0.8845
Epoch 20/20
54/54 [==============================] - 2s 29ms/step - loss: 0.2195 - accuracy:
0.8871 - val_loss: 0.2633 - val_accuracy: 0.8874
```

[22]:
```
# To answer question 13
model1.summary()
model5.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                1860

 dense_1 (Dense)             (None, 20)                420

 dense_2 (Dense)             (None, 1)                 21

=================================================================
Total params: 2,301
Trainable params: 2,301
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 50)                4650

 dense_15 (Dense)            (None, 50)                2550

 dense_16 (Dense)            (None, 50)                2550

 dense_17 (Dense)            (None, 50)                2550

 dense_18 (Dense)            (None, 1)                 51

=================================================================
Total params: 12,351
```

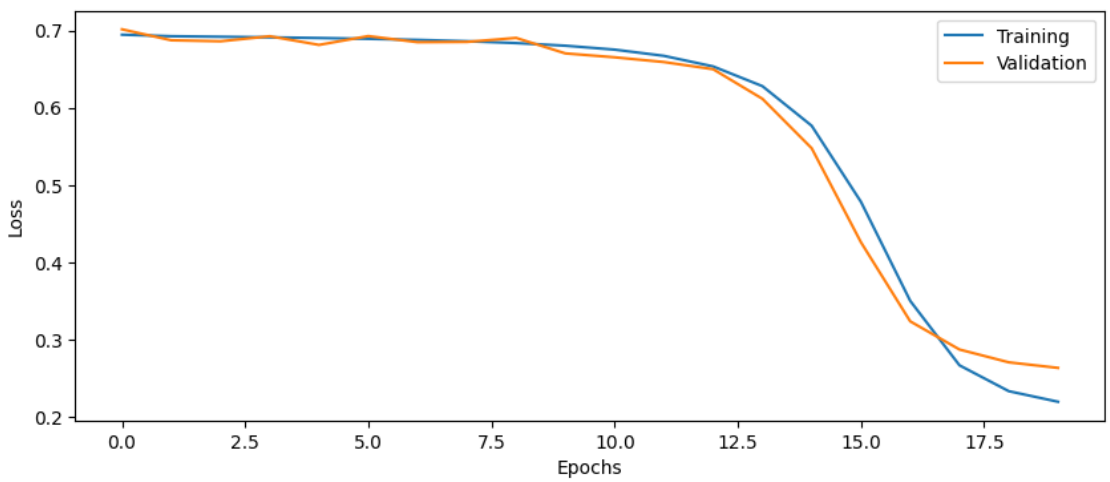
```
Trainable params: 12,351
Non-trainable params: 0
_________________________________________________________________
```

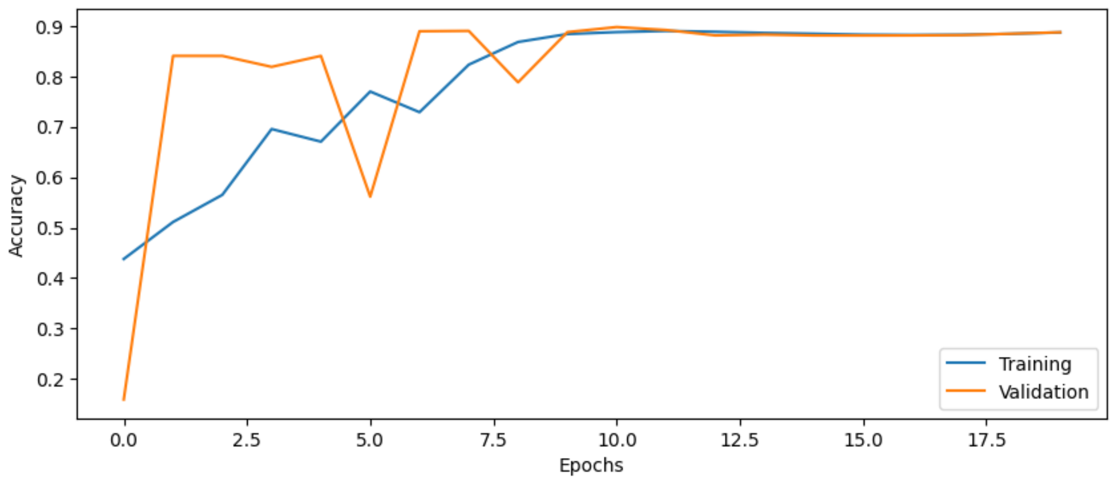
Answer: **Question 13:** The network with 2 Dense layers and 20 nodes per layer have 2301 trainable parameters. The network with 4 Dense layers with 50 nodes each have 12351 trainable parameters.

[23]:
```
# Evaluate model on test data
score = model5.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 4s 1ms/step - loss: 0.2609 -
accuracy: 0.8883
Test loss: 0.2609
Test accuracy: 0.8883
```

[24]:
```
plot_results(history5)
```





16 Part 15: Batch normalization

Now add batch normalization after each dense layer in `build_DNN`. Remember to import BatchNormalization from keras.layers.

See https://keras.io/layers/normalization/ for information about how to call the function.

Question 14: Why is batch normalization important when training deep networks?

Answer:

- **Question 14**: When normalizing the input, the loss function behaves nicer since all values between each layer are normalized. With normalization the training will be more stable and it can also improve speed and performance of the final model.

16.0.1 2 layers, 20 nodes, class weights, batch normalization

[25]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model6 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes = 20,␣
 ↪learning_rate = 0.1, batch_norm = True)

history6 = model6.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
 ↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 2s 16ms/step - loss: 0.2846 - accuracy:
```

```
0.8798 - val_loss: 0.5589 - val_accuracy: 0.8604
Epoch 2/20
54/54 [==============================] - 1s 11ms/step - loss: 0.2112 - accuracy:
0.9072 - val_loss: 0.3915 - val_accuracy: 0.8406
Epoch 3/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1940 - accuracy:
0.9100 - val_loss: 0.3137 - val_accuracy: 0.8407
Epoch 4/20
54/54 [==============================] - 1s 20ms/step - loss: 0.1858 - accuracy:
0.9111 - val_loss: 0.2570 - val_accuracy: 0.8505
Epoch 5/20
54/54 [==============================] - 1s 15ms/step - loss: 0.1808 - accuracy:
0.9123 - val_loss: 0.2118 - val_accuracy: 0.8730
Epoch 6/20
54/54 [==============================] - 1s 13ms/step - loss: 0.1773 - accuracy:
0.9139 - val_loss: 0.1859 - val_accuracy: 0.9073
Epoch 7/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1748 - accuracy:
0.9148 - val_loss: 0.1787 - val_accuracy: 0.9147
Epoch 8/20
54/54 [==============================] - 1s 11ms/step - loss: 0.1729 - accuracy:
0.9151 - val_loss: 0.1851 - val_accuracy: 0.9150
Epoch 9/20
54/54 [==============================] - 1s 11ms/step - loss: 0.1714 - accuracy:
0.9153 - val_loss: 0.1906 - val_accuracy: 0.9151
Epoch 10/20
54/54 [==============================] - 1s 11ms/step - loss: 0.1700 - accuracy:
0.9155 - val_loss: 0.1989 - val_accuracy: 0.9150
Epoch 11/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1689 - accuracy:
0.9157 - val_loss: 0.2020 - val_accuracy: 0.9153
Epoch 12/20
54/54 [==============================] - 1s 11ms/step - loss: 0.1679 - accuracy:
0.9158 - val_loss: 0.2084 - val_accuracy: 0.9150
Epoch 13/20
54/54 [==============================] - 1s 13ms/step - loss: 0.1670 - accuracy:
0.9159 - val_loss: 0.2269 - val_accuracy: 0.9149
Epoch 14/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1663 - accuracy:
0.9161 - val_loss: 0.2068 - val_accuracy: 0.9155
Epoch 15/20
54/54 [==============================] - 1s 9ms/step - loss: 0.1655 - accuracy:
0.9162 - val_loss: 0.2036 - val_accuracy: 0.9159
Epoch 16/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1650 - accuracy:
0.9164 - val_loss: 0.2153 - val_accuracy: 0.9156
Epoch 17/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1642 - accuracy:
```

```
0.9165 - val_loss: 0.2091 - val_accuracy: 0.9161
Epoch 18/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1637 - accuracy:
0.9167 - val_loss: 0.2028 - val_accuracy: 0.9163
Epoch 19/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1632 - accuracy:
0.9168 - val_loss: 0.2042 - val_accuracy: 0.9165
Epoch 20/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1627 - accuracy:
0.9170 - val_loss: 0.2066 - val_accuracy: 0.9165
```

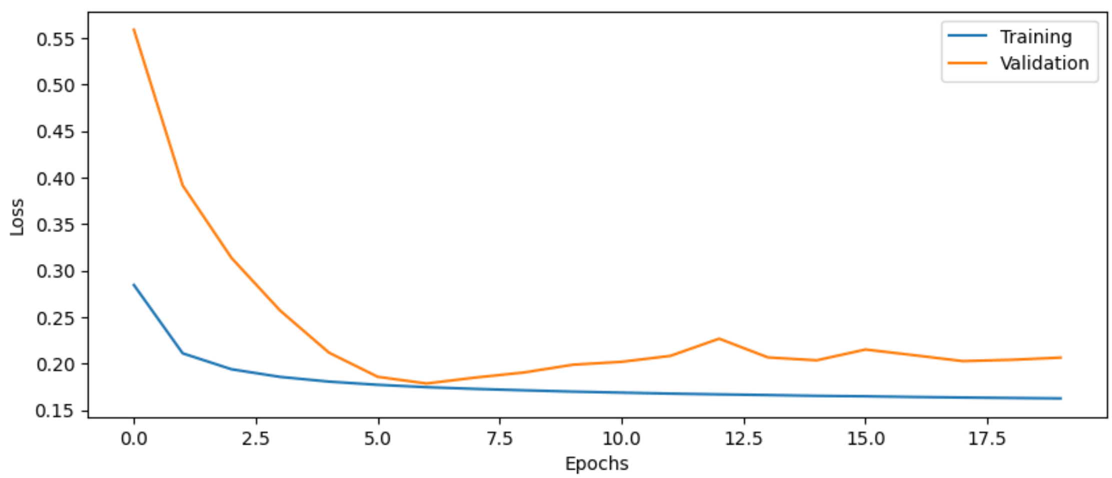
[26]:

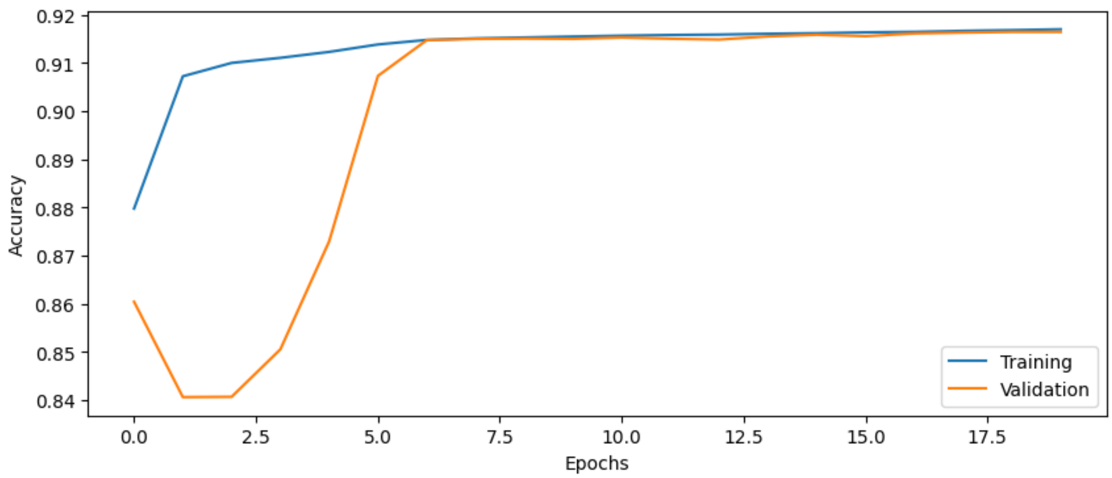
```
# Evaluate model on test data
score = model6.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 4s 1ms/step - loss: 0.2032 -
accuracy: 0.9178
Test loss: 0.2032
Test accuracy: 0.9178
```

[27]:

```
plot_results(history6)
```





17 Part 16: Activation function

Try changing the activation function in each layer from sigmoid to ReLU, write down the test accuracy.

Note: the last layer should still have a sigmoid activation function.

https://keras.io/api/layers/activations/

17.0.1 2 layers, 20 nodes, class weights, ReLU, no batch normalization

[28]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model7 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes = 20,␣
 ↪learning_rate = 0.1, act_fun = 'relu')

history7 = model7.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
 ↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 1s 9ms/step - loss: 0.3063 - accuracy:
0.8799 - val_loss: 0.2803 - val_accuracy: 0.8883
Epoch 2/20
54/54 [==============================] - 0s 7ms/step - loss: 0.2037 - accuracy:
0.8925 - val_loss: 0.2547 - val_accuracy: 0.8942
Epoch 3/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1917 - accuracy:
```

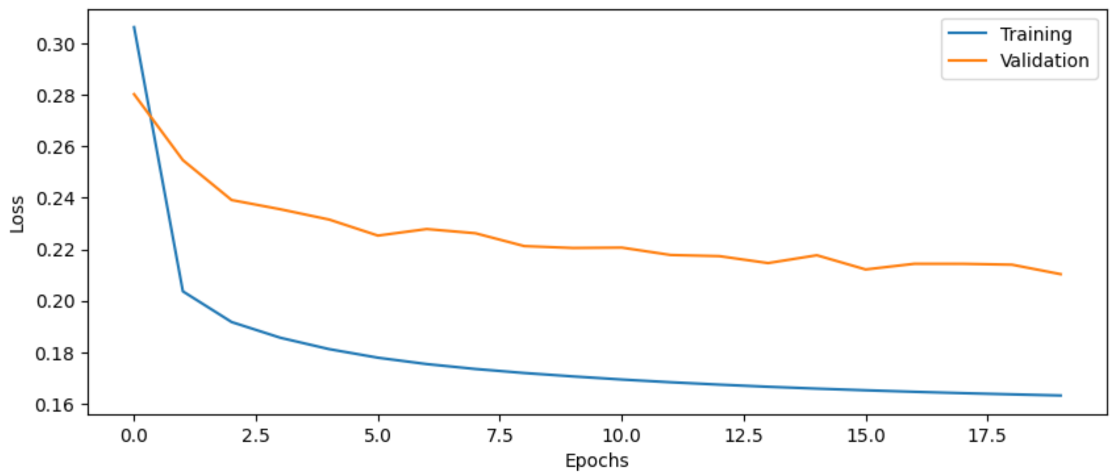
```
0.8982 - val_loss: 0.2391 - val_accuracy: 0.8999
Epoch 4/20
54/54 [==============================] - 0s 8ms/step - loss: 0.1856 - accuracy:
0.9023 - val_loss: 0.2356 - val_accuracy: 0.9034
Epoch 5/20
54/54 [==============================] - 0s 7ms/step - loss: 0.1812 - accuracy:
0.9072 - val_loss: 0.2316 - val_accuracy: 0.9087
Epoch 6/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1779 - accuracy:
0.9104 - val_loss: 0.2253 - val_accuracy: 0.9107
Epoch 7/20
54/54 [==============================] - 0s 8ms/step - loss: 0.1754 - accuracy:
0.9117 - val_loss: 0.2279 - val_accuracy: 0.9112
Epoch 8/20
54/54 [==============================] - 0s 8ms/step - loss: 0.1735 - accuracy:
0.9123 - val_loss: 0.2262 - val_accuracy: 0.9118
Epoch 9/20
54/54 [==============================] - 0s 9ms/step - loss: 0.1719 - accuracy:
0.9127 - val_loss: 0.2212 - val_accuracy: 0.9121
Epoch 10/20
54/54 [==============================] - 1s 10ms/step - loss: 0.1706 - accuracy:
0.9130 - val_loss: 0.2205 - val_accuracy: 0.9123
Epoch 11/20
54/54 [==============================] - 0s 7ms/step - loss: 0.1694 - accuracy:
0.9132 - val_loss: 0.2207 - val_accuracy: 0.9125
Epoch 12/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1683 - accuracy:
0.9134 - val_loss: 0.2178 - val_accuracy: 0.9127
Epoch 13/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1674 - accuracy:
0.9135 - val_loss: 0.2173 - val_accuracy: 0.9128
Epoch 14/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1666 - accuracy:
0.9137 - val_loss: 0.2146 - val_accuracy: 0.9132
Epoch 15/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1659 - accuracy:
0.9142 - val_loss: 0.2177 - val_accuracy: 0.9135
Epoch 16/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1652 - accuracy:
0.9147 - val_loss: 0.2122 - val_accuracy: 0.9142
Epoch 17/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1646 - accuracy:
0.9151 - val_loss: 0.2144 - val_accuracy: 0.9144
Epoch 18/20
54/54 [==============================] - 0s 7ms/step - loss: 0.1641 - accuracy:
0.9154 - val_loss: 0.2144 - val_accuracy: 0.9147
Epoch 19/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1636 - accuracy:
```

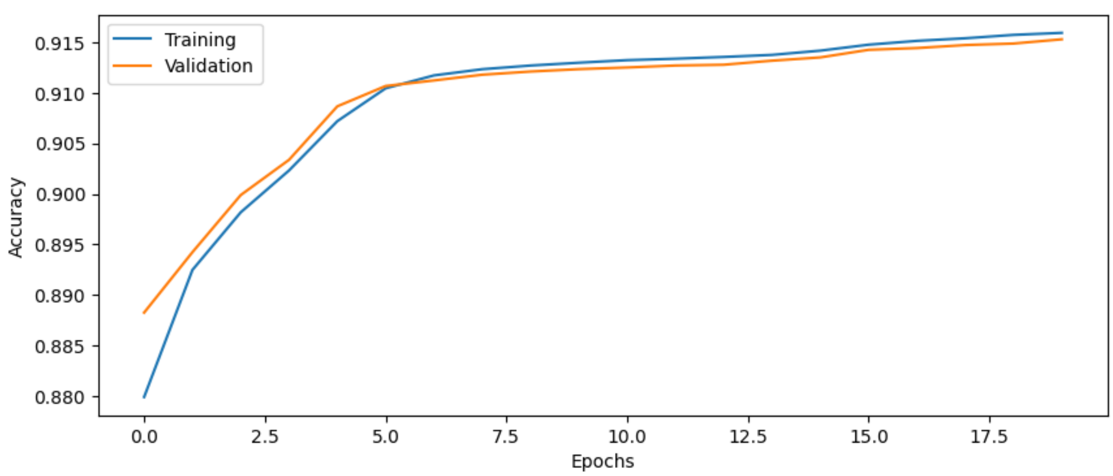
```
0.9157 - val_loss: 0.2140 - val_accuracy: 0.9149
Epoch 20/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1632 - accuracy:
0.9159 - val_loss: 0.2103 - val_accuracy: 0.9153
```

[29]:
```python
# Evaluate model on test data
score = model7.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 3s 962us/step - loss: 0.2070 -
accuracy: 0.9166
Test loss: 0.2070
Test accuracy: 0.9166
```

[30]:
```python
plot_results(history7)
```





18 Part 17: Optimizer

Try changing the optimizer from SGD to Adam (with learning rate 0.1 as before). Remember to import the Adam optimizer from keras.optimizers.

https://keras.io/optimizers/

18.0.1 2 layers, 20 nodes, class weights, Adam optimizer, no batch normalization, sigmoid activations

[31]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model8 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes = 20,␣
↪learning_rate = 0.1, optimizer='Adam')

history8 = model8.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 1s 9ms/step - loss: 0.6012 - accuracy:
0.8758 - val_loss: 0.5179 - val_accuracy: 0.8803
Epoch 2/20
54/54 [==============================] - 0s 6ms/step - loss: 0.4198 - accuracy:
0.8805 - val_loss: 0.3665 - val_accuracy: 0.8788
Epoch 3/20
54/54 [==============================] - 0s 6ms/step - loss: 0.2983 - accuracy:
0.8798 - val_loss: 0.3027 - val_accuracy: 0.8787
Epoch 4/20
54/54 [==============================] - 0s 7ms/step - loss: 0.2498 - accuracy:
0.8799 - val_loss: 0.2816 - val_accuracy: 0.8789
Epoch 5/20
54/54 [==============================] - 0s 7ms/step - loss: 0.2303 - accuracy:
0.8804 - val_loss: 0.2716 - val_accuracy: 0.8800
Epoch 6/20
54/54 [==============================] - 0s 6ms/step - loss: 0.2205 - accuracy:
0.8826 - val_loss: 0.2655 - val_accuracy: 0.8836
Epoch 7/20
54/54 [==============================] - 0s 7ms/step - loss: 0.2144 - accuracy:
0.8856 - val_loss: 0.2616 - val_accuracy: 0.8865
Epoch 8/20
54/54 [==============================] - 0s 6ms/step - loss: 0.2099 - accuracy:
```

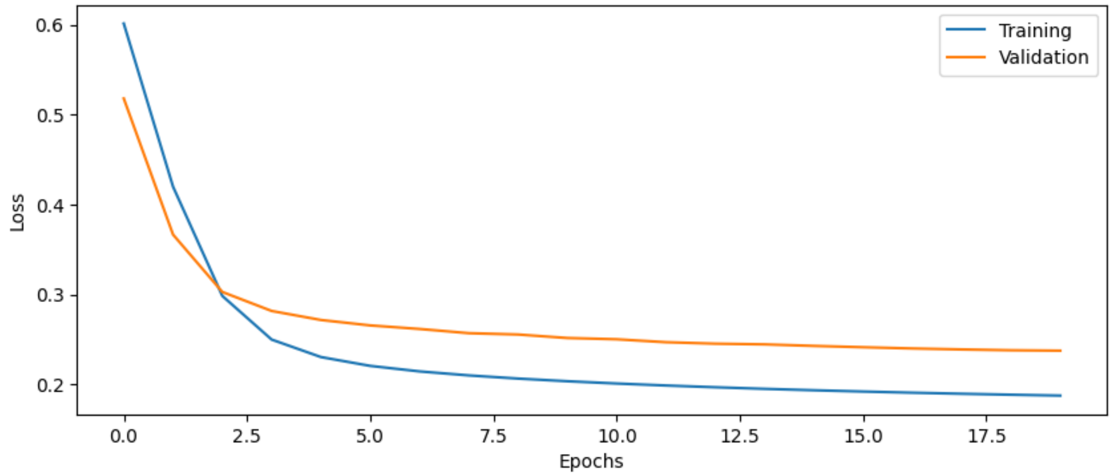
```
0.8884 - val_loss: 0.2569 - val_accuracy: 0.8892
Epoch 9/20
54/54 [==============================] - 0s 7ms/step - loss: 0.2064 - accuracy:
0.8911 - val_loss: 0.2554 - val_accuracy: 0.8914
Epoch 10/20
54/54 [==============================] - 0s 6ms/step - loss: 0.2034 - accuracy:
0.8936 - val_loss: 0.2515 - val_accuracy: 0.8939
Epoch 11/20
54/54 [==============================] - 0s 7ms/step - loss: 0.2009 - accuracy:
0.8954 - val_loss: 0.2501 - val_accuracy: 0.8951
Epoch 12/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1987 - accuracy:
0.8964 - val_loss: 0.2468 - val_accuracy: 0.8959
Epoch 13/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1967 - accuracy:
0.8970 - val_loss: 0.2452 - val_accuracy: 0.8964
Epoch 14/20
54/54 [==============================] - 0s 7ms/step - loss: 0.1950 - accuracy:
0.8976 - val_loss: 0.2444 - val_accuracy: 0.8969
Epoch 15/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1934 - accuracy:
0.8982 - val_loss: 0.2427 - val_accuracy: 0.8975
Epoch 16/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1920 - accuracy:
0.8987 - val_loss: 0.2412 - val_accuracy: 0.8980
Epoch 17/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1907 - accuracy:
0.8993 - val_loss: 0.2399 - val_accuracy: 0.8987
Epoch 18/20
54/54 [==============================] - 0s 8ms/step - loss: 0.1895 - accuracy:
0.8999 - val_loss: 0.2388 - val_accuracy: 0.8993
Epoch 19/20
54/54 [==============================] - 0s 6ms/step - loss: 0.1884 - accuracy:
0.9006 - val_loss: 0.2379 - val_accuracy: 0.9000
Epoch 20/20
54/54 [==============================] - 0s 7ms/step - loss: 0.1875 - accuracy:
0.9015 - val_loss: 0.2374 - val_accuracy: 0.9007
```

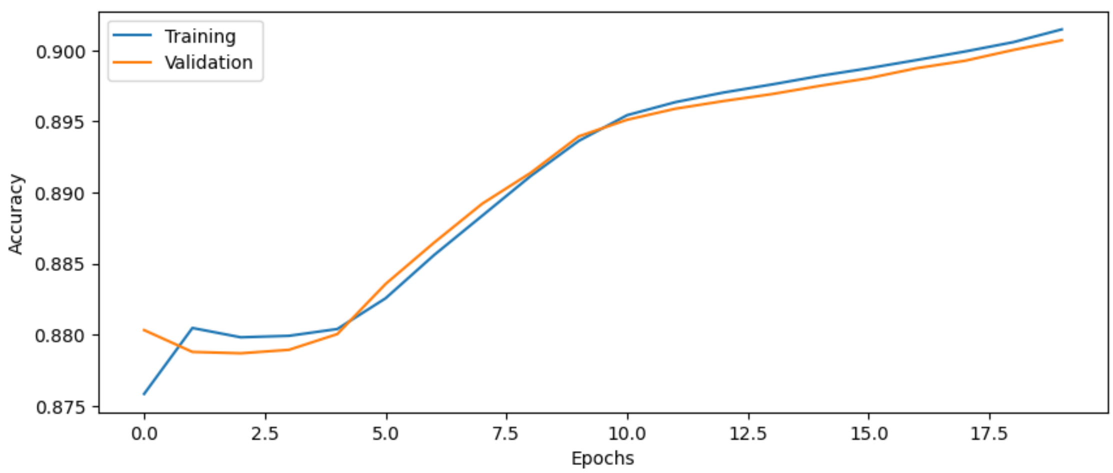
[32]:

```python
# Evaluate model on test data
score = model8.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2339 -
accuracy: 0.9023
Test loss: 0.2339
Test accuracy: 0.9023
```

[33]:
```
plot_results(history8)
```





19 Part 18: Dropout regularization

Dropout is a type of regularization that can improve accuracy for validation and test data. It randomly removes connections to force the neural network to not rely too much on a small number of weights.

Add a Dropout layer after each Dense layer (but not after the final dense layer) in `build_DNN`, with a dropout probability of 50%. Remember to first import the Dropout layer from keras.layers

See https://keras.io/api/layers/regularization_layers/dropout/ for how the Dropout layer works.

Question 15: How does the validation accuracy change when adding dropout?

Question 16: How does the test accuracy change when adding dropout?

19.0.1 2 layers, 20 nodes, class weights, dropout, SGD optimizer, no batch normalization, sigmoid activations

[34]:
```
# Setup some training parameters
batch_size = 10000
epochs = 20
input_shape = 92

# Build and train model
model9 = build_DNN(input_shape = input_shape, n_layers = 2, n_nodes = 20,␣
↪learning_rate = 0.1, dropout = True)

history9 = model9.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/20
54/54 [==============================] - 2s 24ms/step - loss: 0.7097 - accuracy:
0.5151 - val_loss: 0.6315 - val_accuracy: 0.8834
Epoch 2/20
54/54 [==============================] - 1s 16ms/step - loss: 0.6510 - accuracy:
0.6229 - val_loss: 0.5724 - val_accuracy: 0.8870
Epoch 3/20
54/54 [==============================] - 1s 12ms/step - loss: 0.5957 - accuracy:
0.6906 - val_loss: 0.4981 - val_accuracy: 0.8848
Epoch 4/20
54/54 [==============================] - 1s 15ms/step - loss: 0.5316 - accuracy:
0.7408 - val_loss: 0.4130 - val_accuracy: 0.8830
Epoch 5/20
54/54 [==============================] - 1s 10ms/step - loss: 0.4690 - accuracy:
0.7789 - val_loss: 0.3467 - val_accuracy: 0.8810
Epoch 6/20
54/54 [==============================] - 1s 10ms/step - loss: 0.4174 - accuracy:
0.8085 - val_loss: 0.3074 - val_accuracy: 0.8799
Epoch 7/20
54/54 [==============================] - 1s 11ms/step - loss: 0.3821 - accuracy:
0.8263 - val_loss: 0.2898 - val_accuracy: 0.8792
Epoch 8/20
54/54 [==============================] - 1s 14ms/step - loss: 0.3560 - accuracy:
0.8385 - val_loss: 0.2808 - val_accuracy: 0.8792
Epoch 9/20
54/54 [==============================] - 1s 13ms/step - loss: 0.3377 - accuracy:
0.8475 - val_loss: 0.2756 - val_accuracy: 0.8792
Epoch 10/20
```

```
54/54 [==============================] - 1s 14ms/step - loss: 0.3236 - accuracy:
0.8535 - val_loss: 0.2735 - val_accuracy: 0.8793
Epoch 11/20
54/54 [==============================] - 1s 13ms/step - loss: 0.3127 - accuracy:
0.8583 - val_loss: 0.2734 - val_accuracy: 0.8794
Epoch 12/20
54/54 [==============================] - 0s 9ms/step - loss: 0.3041 - accuracy:
0.8613 - val_loss: 0.2723 - val_accuracy: 0.8796
Epoch 13/20
54/54 [==============================] - 1s 17ms/step - loss: 0.2960 - accuracy:
0.8642 - val_loss: 0.2699 - val_accuracy: 0.8799
Epoch 14/20
54/54 [==============================] - 1s 12ms/step - loss: 0.2908 - accuracy:
0.8664 - val_loss: 0.2696 - val_accuracy: 0.8803
Epoch 15/20
54/54 [==============================] - 1s 13ms/step - loss: 0.2853 - accuracy:
0.8678 - val_loss: 0.2688 - val_accuracy: 0.8807
Epoch 16/20
54/54 [==============================] - 1s 10ms/step - loss: 0.2801 - accuracy:
0.8697 - val_loss: 0.2682 - val_accuracy: 0.8813
Epoch 17/20
54/54 [==============================] - 1s 14ms/step - loss: 0.2769 - accuracy:
0.8706 - val_loss: 0.2677 - val_accuracy: 0.8822
Epoch 18/20
54/54 [==============================] - 1s 13ms/step - loss: 0.2715 - accuracy:
0.8719 - val_loss: 0.2662 - val_accuracy: 0.8834
Epoch 19/20
54/54 [==============================] - 1s 14ms/step - loss: 0.2693 - accuracy:
0.8728 - val_loss: 0.2653 - val_accuracy: 0.8841
Epoch 20/20
54/54 [==============================] - 1s 11ms/step - loss: 0.2649 - accuracy:
0.8741 - val_loss: 0.2632 - val_accuracy: 0.8851
```

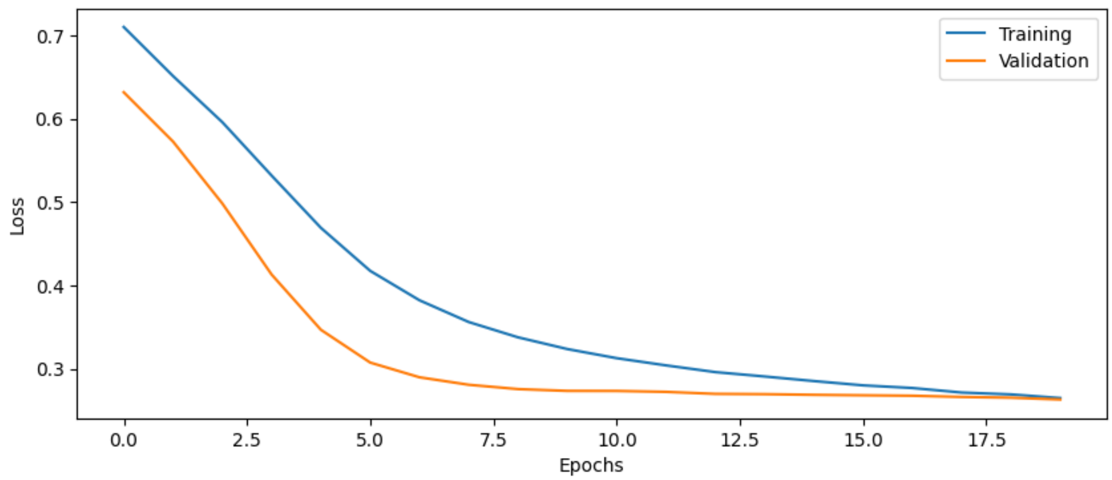
[35]:

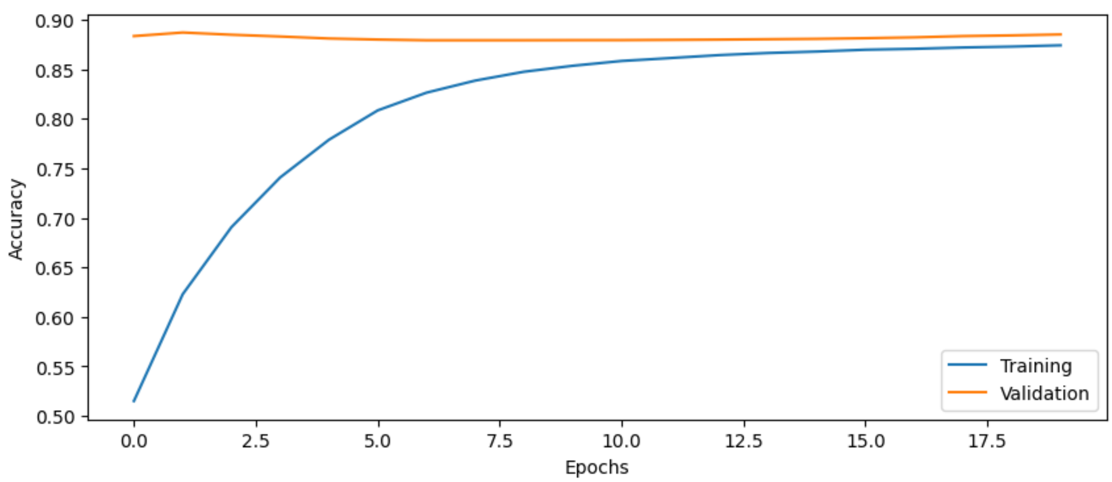
```python
# Evaluate model on test data
score = model9.evaluate(Xtest, Ytest)

print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 4s 1ms/step - loss: 0.2605 -
accuracy: 0.8858
Test loss: 0.2605
Test accuracy: 0.8858
```

[36]:

```python
plot_results(history9)
```





Answer:

- **Question 15**: The validation accuracy does not change a lot when training for longer epochs. This is because the model is regularized by the dropout layers and is better at generalizing to new data.
- **Question 16**: The test accuracy is worse than the models where we added Batch normalization, changed activiation function to relu, and the model with Adam as optimizer.

20 Part 19: Improving performance

Spend some time (30 - 90 minutes) playing with the network architecture (number of layers, number of nodes per layer, activation function) and other hyper parameters (optimizer, learning rate, batch size, number of epochs, degree of regularization). For example, try a much deeper network. How much does the training time increase for a network with 10 layers?

Question 17: How high classification accuracy can you achieve for the test data? What is your best configuration?

[37]:
```python
# Find your best configuration for the DNN
batch_size = 10000
epochs = 40
input_shape = 92

# Build and train DNN
model10 = build_DNN(input_shape = input_shape, n_layers = 5, n_nodes = 50,␣
↪act_fun='relu', learning_rate = 0.1, optimizer='Adam', batch_norm=True)


history10 = model10.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/40
54/54 [==============================] - 4s 48ms/step - loss: 0.2864 - accuracy:
0.8838 - val_loss: 0.3934 - val_accuracy: 0.8410
Epoch 2/40
54/54 [==============================] - 3s 50ms/step - loss: 0.2027 - accuracy:
0.9109 - val_loss: 0.3083 - val_accuracy: 0.8429
Epoch 3/40
54/54 [==============================] - 2s 35ms/step - loss: 0.1843 - accuracy:
0.9134 - val_loss: 0.2449 - val_accuracy: 0.8598
Epoch 4/40
54/54 [==============================] - 2s 37ms/step - loss: 0.1760 - accuracy:
0.9149 - val_loss: 0.1889 - val_accuracy: 0.8894
Epoch 5/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1710 - accuracy:
0.9158 - val_loss: 0.1659 - val_accuracy: 0.9181
Epoch 6/40
54/54 [==============================] - 2s 39ms/step - loss: 0.1675 - accuracy:
0.9164 - val_loss: 0.1680 - val_accuracy: 0.9178
Epoch 7/40
54/54 [==============================] - 2s 36ms/step - loss: 0.1648 - accuracy:
0.9170 - val_loss: 0.1787 - val_accuracy: 0.9183
Epoch 8/40
54/54 [==============================] - 2s 38ms/step - loss: 0.1632 - accuracy:
0.9174 - val_loss: 0.1862 - val_accuracy: 0.9178
Epoch 9/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1614 - accuracy:
0.9178 - val_loss: 0.1928 - val_accuracy: 0.9175
Epoch 10/40
```

```
54/54 [==============================] - 2s 35ms/step - loss: 0.1596 - accuracy:
0.9182 - val_loss: 0.2026 - val_accuracy: 0.9174
Epoch 11/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1586 - accuracy:
0.9185 - val_loss: 0.1995 - val_accuracy: 0.9176
Epoch 12/40
54/54 [==============================] - 2s 32ms/step - loss: 0.1573 - accuracy:
0.9188 - val_loss: 0.2004 - val_accuracy: 0.9173
Epoch 13/40
54/54 [==============================] - 2s 32ms/step - loss: 0.1560 - accuracy:
0.9190 - val_loss: 0.1873 - val_accuracy: 0.9182
Epoch 14/40
54/54 [==============================] - 2s 31ms/step - loss: 0.1546 - accuracy:
0.9196 - val_loss: 0.1820 - val_accuracy: 0.9200
Epoch 15/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1535 - accuracy:
0.9201 - val_loss: 0.2190 - val_accuracy: 0.9181
Epoch 16/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1522 - accuracy:
0.9207 - val_loss: 0.1885 - val_accuracy: 0.9197
Epoch 17/40
54/54 [==============================] - 2s 38ms/step - loss: 0.1510 - accuracy:
0.9213 - val_loss: 0.2047 - val_accuracy: 0.9188
Epoch 18/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1499 - accuracy:
0.9218 - val_loss: 0.1749 - val_accuracy: 0.9224
Epoch 19/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1490 - accuracy:
0.9225 - val_loss: 0.1720 - val_accuracy: 0.9214
Epoch 20/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1473 - accuracy:
0.9232 - val_loss: 0.1690 - val_accuracy: 0.9242
Epoch 21/40
54/54 [==============================] - 2s 35ms/step - loss: 0.1472 - accuracy:
0.9235 - val_loss: 0.1932 - val_accuracy: 0.9214
Epoch 22/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1454 - accuracy:
0.9244 - val_loss: 0.1804 - val_accuracy: 0.9246
Epoch 23/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1443 - accuracy:
0.9251 - val_loss: 0.1816 - val_accuracy: 0.9245
Epoch 24/40
54/54 [==============================] - 2s 37ms/step - loss: 0.1433 - accuracy:
0.9259 - val_loss: 0.1644 - val_accuracy: 0.9265
Epoch 25/40
54/54 [==============================] - 2s 35ms/step - loss: 0.1428 - accuracy:
0.9262 - val_loss: 0.1822 - val_accuracy: 0.9279
Epoch 26/40
```

```
54/54 [==============================] - 2s 35ms/step - loss: 0.1408 - accuracy:
0.9272 - val_loss: 0.2282 - val_accuracy: 0.9187
Epoch 27/40
54/54 [==============================] - 2s 42ms/step - loss: 0.1394 - accuracy:
0.9278 - val_loss: 0.1991 - val_accuracy: 0.9224
Epoch 28/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1397 - accuracy:
0.9279 - val_loss: 0.1747 - val_accuracy: 0.9287
Epoch 29/40
54/54 [==============================] - 2s 32ms/step - loss: 0.1385 - accuracy:
0.9286 - val_loss: 0.1702 - val_accuracy: 0.9291
Epoch 30/40
54/54 [==============================] - 2s 32ms/step - loss: 0.1376 - accuracy:
0.9292 - val_loss: 0.2395 - val_accuracy: 0.9192
Epoch 31/40
54/54 [==============================] - 2s 35ms/step - loss: 0.1371 - accuracy:
0.9294 - val_loss: 0.1709 - val_accuracy: 0.9281
Epoch 32/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1366 - accuracy:
0.9298 - val_loss: 0.1590 - val_accuracy: 0.9320
Epoch 33/40
54/54 [==============================] - 2s 34ms/step - loss: 0.1365 - accuracy:
0.9298 - val_loss: 0.1819 - val_accuracy: 0.9283
Epoch 34/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1346 - accuracy:
0.9305 - val_loss: 0.1606 - val_accuracy: 0.9312
Epoch 35/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1335 - accuracy:
0.9311 - val_loss: 0.1767 - val_accuracy: 0.9307
Epoch 36/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1332 - accuracy:
0.9312 - val_loss: 0.1932 - val_accuracy: 0.9257
Epoch 37/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1341 - accuracy:
0.9307 - val_loss: 0.1827 - val_accuracy: 0.9307
Epoch 38/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1312 - accuracy:
0.9319 - val_loss: 0.2470 - val_accuracy: 0.9196
Epoch 39/40
54/54 [==============================] - 2s 38ms/step - loss: 0.1322 - accuracy:
0.9317 - val_loss: 0.1956 - val_accuracy: 0.9270
Epoch 40/40
54/54 [==============================] - 2s 33ms/step - loss: 0.1315 - accuracy:
0.9320 - val_loss: 0.1885 - val_accuracy: 0.9265
```

[38]:
```
# Evaluate DNN on test data
score = model10.evaluate(Xtest, Ytest)
```

```
print('Test loss: %.4f' % score[0])
print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 4s 1ms/step - loss: 0.1854 -
accuracy: 0.9278
Test loss: 0.1854
Test accuracy: 0.9278
```

Answer:

- **Question 17**: From the previous models, we found that using activation relu, batch normalization, class weights and optimizer Adam improved the models. We decided to also train the model for longer and our model is better than any previous models. The model have an accuracy around 93% on test data.

21 Part 20: Dropout uncertainty

Dropout can also be used during testing, to obtain an estimate of the model uncertainty. Since dropout will randomly remove connections, the network will produce different results every time the same (test) data is put into the network. This technique is called Monte Carlo dropout. For more information, see this paper http://proceedings.mlr.press/v48/gal16.pdf

To achieve this, we need to redefine the Keras Dropout call by running the cell below, and use 'myDropout' in each call to Dropout, in the cell that defines the DNN. The `build_DNN` function takes two boolean arguments, use_dropout and use_custom_dropout, add a standard Dropout layer if use_dropout is true, add a myDropout layer if use_custom_dropout is true.

Run the same test data through the trained network 100 times, with dropout turned on.

Question 18: What is the mean and the standard deviation of the test accuracy?

[39]:
```
import keras.backend as K
import keras

class myDropout(keras.layers.Dropout):
    """Applies Dropout to the input.
    Dropout consists in randomly setting
    a fraction `rate` of input units to 0 at each update during training time,
    which helps prevent overfitting.
    # Arguments
        rate: float between 0 and 1. Fraction of the input units to drop.
        noise_shape: 1D integer tensor representing the shape of the
            binary dropout mask that will be multiplied with the input.
            For instance, if your inputs have shape
            `(batch_size, timesteps, features)` and
            you want the dropout mask to be the same for all timesteps,
            you can use `noise_shape=(batch_size, 1, features)`.
        seed: A Python integer to use as random seed.
```

```
    # References
        - [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](
           http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf)
    """
    def __init__(self, rate, training=True, noise_shape=None, seed=None,␣
↪**kwargs):
        super(myDropout, self).__init__(rate, noise_shape=None,␣
↪seed=None,**kwargs)
        self.training = training


    def call(self, inputs, training=None):
        if 0. < self.rate < 1.:
            noise_shape = self._get_noise_shape(inputs)

            def dropped_inputs():
                return K.dropout(inputs, self.rate, noise_shape,
                                 seed=self.seed)
            if not training:
                return K.in_train_phase(dropped_inputs, inputs, training=self.
↪training)
            return K.in_train_phase(dropped_inputs, inputs, training=training)
        return inputs
```

21.0.1 Your best config, custom dropout

[40]:
```
# Your best training parameters
batch_size = 10000
epochs = 40
input_shape = 92

# Build and train model
model11 = build_DNN(input_shape = input_shape, n_layers = 5, n_nodes = 50,␣
↪act_fun='relu',
                    learning_rate = 0.1, optimizer='Adam', batch_norm=True,␣
↪mydropout = True)


history11 = model11.fit(Xtrain, Ytrain, validation_data = (Xval, Yval),␣
↪class_weight = class_weights, batch_size = batch_size, epochs = epochs)
```

```
Epoch 1/40
54/54 [==============================] - 5s 71ms/step - loss: 0.5669 - accuracy:
0.7605 - val_loss: 0.5205 - val_accuracy: 0.8718
Epoch 2/40
54/54 [==============================] - 3s 49ms/step - loss: 0.3945 - accuracy:
```

```
0.8616 - val_loss: 0.3726 - val_accuracy: 0.8808
Epoch 3/40
54/54 [==============================] - 3s 50ms/step - loss: 0.3402 - accuracy:
0.8743 - val_loss: 0.3269 - val_accuracy: 0.8860
Epoch 4/40
54/54 [==============================] - 2s 46ms/step - loss: 0.3100 - accuracy:
0.8803 - val_loss: 0.3123 - val_accuracy: 0.8867
Epoch 5/40
54/54 [==============================] - 3s 49ms/step - loss: 0.2902 - accuracy:
0.8840 - val_loss: 0.3023 - val_accuracy: 0.8894
Epoch 6/40
54/54 [==============================] - 3s 52ms/step - loss: 0.2762 - accuracy:
0.8865 - val_loss: 0.2980 - val_accuracy: 0.8906
Epoch 7/40
54/54 [==============================] - 3s 54ms/step - loss: 0.2659 - accuracy:
0.8886 - val_loss: 0.2936 - val_accuracy: 0.8920
Epoch 8/40
54/54 [==============================] - 2s 46ms/step - loss: 0.2585 - accuracy:
0.8906 - val_loss: 0.2908 - val_accuracy: 0.8929
Epoch 9/40
54/54 [==============================] - 3s 47ms/step - loss: 0.2500 - accuracy:
0.8930 - val_loss: 0.2882 - val_accuracy: 0.8951
Epoch 10/40
54/54 [==============================] - 3s 49ms/step - loss: 0.2452 - accuracy:
0.8942 - val_loss: 0.2797 - val_accuracy: 0.8968
Epoch 11/40
54/54 [==============================] - 3s 60ms/step - loss: 0.2402 - accuracy:
0.8958 - val_loss: 0.2806 - val_accuracy: 0.8977
Epoch 12/40
54/54 [==============================] - 3s 64ms/step - loss: 0.2350 - accuracy:
0.8974 - val_loss: 0.2755 - val_accuracy: 0.8994
Epoch 13/40
54/54 [==============================] - 3s 49ms/step - loss: 0.2316 - accuracy:
0.8988 - val_loss: 0.2726 - val_accuracy: 0.9008
Epoch 14/40
54/54 [==============================] - 3s 50ms/step - loss: 0.2287 - accuracy:
0.9001 - val_loss: 0.2681 - val_accuracy: 0.9017
Epoch 15/40
54/54 [==============================] - 3s 50ms/step - loss: 0.2258 - accuracy:
0.9010 - val_loss: 0.2652 - val_accuracy: 0.9031
Epoch 16/40
54/54 [==============================] - 3s 55ms/step - loss: 0.2224 - accuracy:
0.9023 - val_loss: 0.2620 - val_accuracy: 0.9040
Epoch 17/40
54/54 [==============================] - 3s 47ms/step - loss: 0.2197 - accuracy:
0.9036 - val_loss: 0.2607 - val_accuracy: 0.9044
Epoch 18/40
54/54 [==============================] - 3s 48ms/step - loss: 0.2181 - accuracy:
```

```
0.9041 - val_loss: 0.2592 - val_accuracy: 0.9049
Epoch 19/40
54/54 [==============================] - 3s 50ms/step - loss: 0.2164 - accuracy:
0.9048 - val_loss: 0.2588 - val_accuracy: 0.9058
Epoch 20/40
54/54 [==============================] - 3s 49ms/step - loss: 0.2144 - accuracy:
0.9052 - val_loss: 0.2555 - val_accuracy: 0.9063
Epoch 21/40
54/54 [==============================] - 3s 48ms/step - loss: 0.2127 - accuracy:
0.9059 - val_loss: 0.2559 - val_accuracy: 0.9066
Epoch 22/40
54/54 [==============================] - 3s 49ms/step - loss: 0.2105 - accuracy:
0.9065 - val_loss: 0.2556 - val_accuracy: 0.9070
Epoch 23/40
54/54 [==============================] - 3s 47ms/step - loss: 0.2100 - accuracy:
0.9068 - val_loss: 0.2543 - val_accuracy: 0.9075
Epoch 24/40
54/54 [==============================] - 2s 46ms/step - loss: 0.2080 - accuracy:
0.9076 - val_loss: 0.2549 - val_accuracy: 0.9073
Epoch 25/40
54/54 [==============================] - 3s 50ms/step - loss: 0.2065 - accuracy:
0.9077 - val_loss: 0.2514 - val_accuracy: 0.9086
Epoch 26/40
54/54 [==============================] - 3s 47ms/step - loss: 0.2063 - accuracy:
0.9080 - val_loss: 0.2488 - val_accuracy: 0.9092
Epoch 27/40
54/54 [==============================] - 3s 47ms/step - loss: 0.2048 - accuracy:
0.9086 - val_loss: 0.2521 - val_accuracy: 0.9087
Epoch 28/40
54/54 [==============================] - 3s 51ms/step - loss: 0.2036 - accuracy:
0.9087 - val_loss: 0.2465 - val_accuracy: 0.9096
Epoch 29/40
54/54 [==============================] - 3s 54ms/step - loss: 0.2025 - accuracy:
0.9091 - val_loss: 0.2481 - val_accuracy: 0.9093
Epoch 30/40
54/54 [==============================] - 3s 46ms/step - loss: 0.2022 - accuracy:
0.9092 - val_loss: 0.2464 - val_accuracy: 0.9100
Epoch 31/40
54/54 [==============================] - 3s 62ms/step - loss: 0.2009 - accuracy:
0.9096 - val_loss: 0.2485 - val_accuracy: 0.9098
Epoch 32/40
54/54 [==============================] - 2s 45ms/step - loss: 0.1999 - accuracy:
0.9097 - val_loss: 0.2453 - val_accuracy: 0.9101
Epoch 33/40
54/54 [==============================] - 2s 44ms/step - loss: 0.2000 - accuracy:
0.9099 - val_loss: 0.2404 - val_accuracy: 0.9102
Epoch 34/40
54/54 [==============================] - 3s 47ms/step - loss: 0.1984 - accuracy:
```

```
0.9101 - val_loss: 0.2455 - val_accuracy: 0.9104
Epoch 35/40
54/54 [==============================] - 2s 46ms/step - loss: 0.1987 - accuracy:
0.9101 - val_loss: 0.2432 - val_accuracy: 0.9106
Epoch 36/40
54/54 [==============================] - 2s 45ms/step - loss: 0.1977 - accuracy:
0.9106 - val_loss: 0.2413 - val_accuracy: 0.9107
Epoch 37/40
54/54 [==============================] - 2s 45ms/step - loss: 0.1971 - accuracy:
0.9107 - val_loss: 0.2446 - val_accuracy: 0.9105
Epoch 38/40
54/54 [==============================] - 2s 45ms/step - loss: 0.1966 - accuracy:
0.9105 - val_loss: 0.2417 - val_accuracy: 0.9107
Epoch 39/40
54/54 [==============================] - 2s 46ms/step - loss: 0.1954 - accuracy:
0.9109 - val_loss: 0.2395 - val_accuracy: 0.9112
Epoch 40/40
54/54 [==============================] - 2s 44ms/step - loss: 0.1949 - accuracy:
0.9113 - val_loss: 0.2423 - val_accuracy: 0.9115
```

[41]:

```python
# Run this cell a few times to evalute the model on test data,
# if you get slightly different test accuracy every time, Dropout during␣
↪testing is working

# Evaluate model on test data
score = model11.evaluate(Xtest, Ytest)

print('Test accuracy: %.4f' % score[1])
```

```
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2396 -
accuracy: 0.9126
Test accuracy: 0.9126
```

[42]:

```python
# Run the testing 100 times, and save the accuracies in an array
scores = []
for iteration in range(0,99):
    score = model11.evaluate(Xtest, Ytest)
    scores.append(score[1])

scores = np.array(scores)

# Calculate and print mean and std of accuracies
print(f"The mean is: {np.mean(scores)}")
print(f"The sd is: {np.std(scores)}")
```

```
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2379 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2388 -
```

```
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2384 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2380 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2383 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2377 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2392 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2383 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2390 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2393 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2384 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2385 -
accuracy: 0.9122
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2390 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2391 -
accuracy: 0.9124
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2393 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2374 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2374 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2387 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2385 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2377 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2380 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2379 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2386 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2378 -
```

```
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2381 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2376 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2387 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2385 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2376 -
accuracy: 0.9128
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2381 -
accuracy: 0.9129
3582/3582 [==============================] - 7s 2ms/step - loss: 0.2376 -
accuracy: 0.9128
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2386 -
accuracy: 0.9125
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2376 -
accuracy: 0.9127
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2391 -
accuracy: 0.9126
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2371 -
accuracy: 0.9128
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2388 -
accuracy: 0.9124
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2380 -
accuracy: 0.9126
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2384 -
accuracy: 0.9127
3582/3582 [==============================] - 7s 2ms/step - loss: 0.2388 -
accuracy: 0.9124
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2375 -
accuracy: 0.9127
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2383 -
accuracy: 0.9126
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2380 -
accuracy: 0.9127
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2378 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2389 -
accuracy: 0.9124
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2389 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2394 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 2ms/step - loss: 0.2396 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2383 -
```

```
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2384 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2384 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2387 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2376 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2372 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2373 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2377 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2380 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2380 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2390 -
accuracy: 0.9123
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2390 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2387 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2376 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2388 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2390 -
accuracy: 0.9126
3582/3582 [==============================] - 6s 2ms/step - loss: 0.2379 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2378 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2389 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2374 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2365 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2380 -
accuracy: 0.9124
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2388 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2385 -
```

```
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2380 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2386 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2381 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2383 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2371 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2381 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2373 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2372 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9122
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2387 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2386 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2371 -
accuracy: 0.9129
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2375 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2374 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2381 -
accuracy: 0.9125
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2364 -
accuracy: 0.9130
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2381 -
accuracy: 0.9124
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2396 -
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2384 -
accuracy: 0.9128
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2386 -
accuracy: 0.9126
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2382 -
```

```
accuracy: 0.9127
3582/3582 [==============================] - 5s 1ms/step - loss: 0.2370 -
accuracy: 0.9126
The mean is: 0.9126587480005591
The sd is: 0.00014974259809960187
```

Answer:

- **Question 18**: From the output the mean of the test accuracy is around 0.9127 and the standard deviation is around 0.0001.

22 Part 21: Cross validation uncertainty

Cross validation (CV) is often used to evaluate a model, by training and testing using different subsets of the data it is possible to get the uncertainty as the standard deviation over folds. We here use a help function from scikit-learn to setup the CV, see https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html . Use 10 folds with shuffling, random state 1234.

Note: We here assume that you have found the best hyper parameters, so here the data are only split into training and testing, no validation.

Question 19: What is the mean and the standard deviation of the test accuracy?

Question 20: What is the main advantage of dropout compared to CV for estimating test uncertainty? The difference may not be so large in this notebook, but imagine that you have a network that takes 24 hours to train.

[44]:
```python
from sklearn.model_selection import StratifiedKFold

test_acc = []

# Define 10-fold cross validation
skf = StratifiedKFold(n_splits=10, random_state=1234, shuffle=True)

# Loop over cross validation folds
for i, (train_index, test_index) in enumerate(skf.split(X, Y)):
    Xtrain = X[train_index, ]
    Ytrain = Y[train_index]
    Xtest = X[test_index, ]
    Ytest = Y[test_index]

    # Calculate class weights for current split
    weights = class_weight.compute_class_weight(class_weight = "balanced",␣
 ↪classes = np.unique(Ytrain), y = Ytrain)
    class_weights = {0: weights[0],
                     1: weights[1]}
```

```
    # Rebuild the DNN model, to not continue training on the previously trained␣
↪model
    model = build_DNN(input_shape = input_shape,
                      n_layers = 5,
                      n_nodes = 50,
                      act_fun = 'relu',
                      learning_rate = 0.1,
                      optimizer = 'Adam',
                      batch_norm = True,
                      mydropout = True)

    # Fit the model with training set and class weights for this fold
    history = model.fit(Xtrain, Ytrain,
                        class_weight = class_weights,
                        batch_size = batch_size,
                        epochs = epochs)

    # Evaluate the model using the test set for this fold
    score = model.evaluate(Xtest, Ytest)

    # Save the test accuracy in an array
    test_acc.append(score[1])

# Calculate and print mean and std of accuracies
scores = np.array(test_acc)
print(f"The mean is: {np.mean(scores)}")
print(f"The sd is: {np.std(scores)}")
```

```
Epoch 1/40
69/69 [==============================] - 6s 54ms/step - loss: 0.5233 - accuracy:
0.8022
Epoch 2/40
69/69 [==============================] - 3s 45ms/step - loss: 0.3687 - accuracy:
0.8713
Epoch 3/40
69/69 [==============================] - 4s 53ms/step - loss: 0.3200 - accuracy:
0.8795
Epoch 4/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2934 - accuracy:
0.8851
Epoch 5/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2768 - accuracy:
0.8891
Epoch 6/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2640 - accuracy:
0.8924
Epoch 7/40
```

```
69/69 [==============================] - 3s 40ms/step - loss: 0.2534 - accuracy:
0.8951
Epoch 8/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2451 - accuracy:
0.8974
Epoch 9/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2386 - accuracy:
0.8991
Epoch 10/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2318 - accuracy:
0.9011
Epoch 11/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2280 - accuracy:
0.9021
Epoch 12/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2227 - accuracy:
0.9038
Epoch 13/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2197 - accuracy:
0.9045
Epoch 14/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2171 - accuracy:
0.9054
Epoch 15/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2144 - accuracy:
0.9064
Epoch 16/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2124 - accuracy:
0.9070
Epoch 17/40
69/69 [==============================] - 3s 39ms/step - loss: 0.2101 - accuracy:
0.9079
Epoch 18/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2080 - accuracy:
0.9082
Epoch 19/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2065 - accuracy:
0.9085
Epoch 20/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2052 - accuracy:
0.9091
Epoch 21/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2041 - accuracy:
0.9095
Epoch 22/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2023 - accuracy:
0.9097
Epoch 23/40
```

```
69/69 [==============================] - 3s 41ms/step - loss: 0.2010 - accuracy:
0.9099
Epoch 24/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2006 - accuracy:
0.9103
Epoch 25/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1994 - accuracy:
0.9104
Epoch 26/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1979 - accuracy:
0.9108
Epoch 27/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1976 - accuracy:
0.9109
Epoch 28/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1968 - accuracy:
0.9110
Epoch 29/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1954 - accuracy:
0.9113
Epoch 30/40
69/69 [==============================] - 3s 39ms/step - loss: 0.1953 - accuracy:
0.9114
Epoch 31/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1940 - accuracy:
0.9116
Epoch 32/40
69/69 [==============================] - 3s 39ms/step - loss: 0.1934 - accuracy:
0.9116
Epoch 33/40
69/69 [==============================] - 3s 39ms/step - loss: 0.1928 - accuracy:
0.9119
Epoch 34/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1919 - accuracy:
0.9119
Epoch 35/40
69/69 [==============================] - 3s 39ms/step - loss: 0.1918 - accuracy:
0.9121
Epoch 36/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1915 - accuracy:
0.9121
Epoch 37/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1907 - accuracy:
0.9123
Epoch 38/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1906 - accuracy:
0.9123
Epoch 39/40
```

```
69/69 [==============================] - 3s 40ms/step - loss: 0.1898 - accuracy:
0.9124
Epoch 40/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1896 - accuracy:
0.9125
2388/2388 [==============================] - 3s 1ms/step - loss: 0.2372 -
accuracy: 0.9122
Epoch 1/40
69/69 [==============================] - 4s 40ms/step - loss: 0.5295 - accuracy:
0.7965
Epoch 2/40
69/69 [==============================] - 3s 40ms/step - loss: 0.3676 - accuracy:
0.8669
Epoch 3/40
69/69 [==============================] - 3s 40ms/step - loss: 0.3177 - accuracy:
0.8766
Epoch 4/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2909 - accuracy:
0.8822
Epoch 5/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2731 - accuracy:
0.8867
Epoch 6/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2603 - accuracy:
0.8902
Epoch 7/40
69/69 [==============================] - 3s 39ms/step - loss: 0.2508 - accuracy:
0.8932
Epoch 8/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2439 - accuracy:
0.8956
Epoch 9/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2374 - accuracy:
0.8974
Epoch 10/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2325 - accuracy:
0.8995
Epoch 11/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2275 - accuracy:
0.9012
Epoch 12/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2246 - accuracy:
0.9023
Epoch 13/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2205 - accuracy:
0.9036
Epoch 14/40
69/69 [==============================] - 3s 39ms/step - loss: 0.2175 - accuracy:
```

```
0.9046
Epoch 15/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2143 - accuracy:
0.9056
Epoch 16/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2125 - accuracy:
0.9061
Epoch 17/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2102 - accuracy:
0.9070
Epoch 18/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2084 - accuracy:
0.9075
Epoch 19/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2073 - accuracy:
0.9080
Epoch 20/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2050 - accuracy:
0.9086
Epoch 21/40
69/69 [==============================] - 3s 46ms/step - loss: 0.2041 - accuracy:
0.9091
Epoch 22/40
69/69 [==============================] - 4s 53ms/step - loss: 0.2024 - accuracy:
0.9095
Epoch 23/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2011 - accuracy:
0.9097
Epoch 24/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2004 - accuracy:
0.9101
Epoch 25/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1992 - accuracy:
0.9102
Epoch 26/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1981 - accuracy:
0.9105
Epoch 27/40
69/69 [==============================] - 3s 48ms/step - loss: 0.1971 - accuracy:
0.9108
Epoch 28/40
69/69 [==============================] - 4s 62ms/step - loss: 0.1965 - accuracy:
0.9111
Epoch 29/40
69/69 [==============================] - 3s 46ms/step - loss: 0.1955 - accuracy:
0.9111
Epoch 30/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1946 - accuracy:
```

```
0.9113
Epoch 31/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1940 - accuracy:
0.9115
Epoch 32/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1938 - accuracy:
0.9116
Epoch 33/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1922 - accuracy:
0.9117
Epoch 34/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1922 - accuracy:
0.9117
Epoch 35/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1915 - accuracy:
0.9119
Epoch 36/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1909 - accuracy:
0.9119
Epoch 37/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1904 - accuracy:
0.9122
Epoch 38/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1895 - accuracy:
0.9121
Epoch 39/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1891 - accuracy:
0.9125
Epoch 40/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1884 - accuracy:
0.9125
2388/2388 [==============================] - 3s 1ms/step - loss: 0.2239 -
accuracy: 0.9155
Epoch 1/40
69/69 [==============================] - 4s 39ms/step - loss: 0.5689 - accuracy:
0.7686
Epoch 2/40
69/69 [==============================] - 3s 39ms/step - loss: 0.3816 - accuracy:
0.8644
Epoch 3/40
69/69 [==============================] - 3s 39ms/step - loss: 0.3284 - accuracy:
0.8755
Epoch 4/40
69/69 [==============================] - 3s 39ms/step - loss: 0.3009 - accuracy:
0.8809
Epoch 5/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2807 - accuracy:
0.8853
```

```
Epoch 6/40
69/69 [==============================] - 4s 58ms/step - loss: 0.2681 - accuracy:
0.8884
Epoch 7/40
69/69 [==============================] - 5s 66ms/step - loss: 0.2576 - accuracy:
0.8910
Epoch 8/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2469 - accuracy:
0.8936
Epoch 9/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2409 - accuracy:
0.8953
Epoch 10/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2351 - accuracy:
0.8970
Epoch 11/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2291 - accuracy:
0.8990
Epoch 12/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2252 - accuracy:
0.9004
Epoch 13/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2217 - accuracy:
0.9017
Epoch 14/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2184 - accuracy:
0.9029
Epoch 15/40
69/69 [==============================] - 4s 51ms/step - loss: 0.2150 - accuracy:
0.9042
Epoch 16/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2135 - accuracy:
0.9050
Epoch 17/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2112 - accuracy:
0.9057
Epoch 18/40
69/69 [==============================] - 4s 53ms/step - loss: 0.2088 - accuracy:
0.9066
Epoch 19/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2069 - accuracy:
0.9075
Epoch 20/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2051 - accuracy:
0.9080
Epoch 21/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2038 - accuracy:
0.9085
```

```
Epoch 22/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2026 - accuracy:
0.9090
Epoch 23/40
69/69 [==============================] - 3s 46ms/step - loss: 0.2014 - accuracy:
0.9096
Epoch 24/40
69/69 [==============================] - 3s 49ms/step - loss: 0.2006 - accuracy:
0.9101
Epoch 25/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1998 - accuracy:
0.9103
Epoch 26/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1985 - accuracy:
0.9106
Epoch 27/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1977 - accuracy:
0.9107
Epoch 28/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1961 - accuracy:
0.9111
Epoch 29/40
69/69 [==============================] - 3s 47ms/step - loss: 0.1956 - accuracy:
0.9114
Epoch 30/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1952 - accuracy:
0.9114
Epoch 31/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1944 - accuracy:
0.9115
Epoch 32/40
69/69 [==============================] - 3s 50ms/step - loss: 0.1939 - accuracy:
0.9116
Epoch 33/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1933 - accuracy:
0.9120
Epoch 34/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1928 - accuracy:
0.9121
Epoch 35/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1918 - accuracy:
0.9122
Epoch 36/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1913 - accuracy:
0.9124
Epoch 37/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1911 - accuracy:
0.9126
```

```
Epoch 38/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1904 - accuracy:
0.9127
Epoch 39/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1903 - accuracy:
0.9127
Epoch 40/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1900 - accuracy:
0.9129
2388/2388 [==============================] - 4s 1ms/step - loss: 0.2359 -
accuracy: 0.9132
Epoch 1/40
69/69 [==============================] - 4s 41ms/step - loss: 0.5221 - accuracy:
0.7827
Epoch 2/40
69/69 [==============================] - 3s 43ms/step - loss: 0.3630 - accuracy:
0.8697
Epoch 3/40
69/69 [==============================] - 3s 43ms/step - loss: 0.3167 - accuracy:
0.8797
Epoch 4/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2894 - accuracy:
0.8849
Epoch 5/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2729 - accuracy:
0.8891
Epoch 6/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2593 - accuracy:
0.8927
Epoch 7/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2489 - accuracy:
0.8955
Epoch 8/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2421 - accuracy:
0.8980
Epoch 9/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2355 - accuracy:
0.8996
Epoch 10/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2298 - accuracy:
0.9014
Epoch 11/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2248 - accuracy:
0.9030
Epoch 12/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2221 - accuracy:
0.9041
Epoch 13/40
```

```
69/69 [==============================] - 3s 48ms/step - loss: 0.2183 - accuracy:
0.9050
Epoch 14/40
69/69 [==============================] - 3s 46ms/step - loss: 0.2154 - accuracy:
0.9058
Epoch 15/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2133 - accuracy:
0.9067
Epoch 16/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2096 - accuracy:
0.9074
Epoch 17/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2083 - accuracy:
0.9077
Epoch 18/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2059 - accuracy:
0.9085
Epoch 19/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2041 - accuracy:
0.9090
Epoch 20/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2027 - accuracy:
0.9092
Epoch 21/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2025 - accuracy:
0.9094
Epoch 22/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2006 - accuracy:
0.9098
Epoch 23/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1997 - accuracy:
0.9103
Epoch 24/40
69/69 [==============================] - 3s 47ms/step - loss: 0.1981 - accuracy:
0.9104
Epoch 25/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1979 - accuracy:
0.9107
Epoch 26/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1965 - accuracy:
0.9108
Epoch 27/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1958 - accuracy:
0.9111
Epoch 28/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1956 - accuracy:
0.9111
Epoch 29/40
```

```
69/69 [==============================] - 3s 44ms/step - loss: 0.1940 - accuracy:
0.9116
Epoch 30/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1939 - accuracy:
0.9116
Epoch 31/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1933 - accuracy:
0.9119
Epoch 32/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1929 - accuracy:
0.9118
Epoch 33/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1910 - accuracy:
0.9120
Epoch 34/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1910 - accuracy:
0.9122
Epoch 35/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1905 - accuracy:
0.9123
Epoch 36/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1908 - accuracy:
0.9123
Epoch 37/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1900 - accuracy:
0.9123
Epoch 38/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1890 - accuracy:
0.9127
Epoch 39/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1892 - accuracy:
0.9127
Epoch 40/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1888 - accuracy:
0.9126
2388/2388 [==============================] - 4s 2ms/step - loss: 0.2389 -
accuracy: 0.9114
Epoch 1/40
69/69 [==============================] - 4s 39ms/step - loss: 0.5371 - accuracy:
0.7683
Epoch 2/40
69/69 [==============================] - 3s 38ms/step - loss: 0.3641 - accuracy:
0.8620
Epoch 3/40
69/69 [==============================] - 3s 44ms/step - loss: 0.3160 - accuracy:
0.8745
Epoch 4/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2883 - accuracy:
```

```
0.8817
Epoch 5/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2716 - accuracy:
0.8866
Epoch 6/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2593 - accuracy:
0.8907
Epoch 7/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2503 - accuracy:
0.8942
Epoch 8/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2421 - accuracy:
0.8972
Epoch 9/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2364 - accuracy:
0.8993
Epoch 10/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2305 - accuracy:
0.9014
Epoch 11/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2267 - accuracy:
0.9030
Epoch 12/40
69/69 [==============================] - 3s 50ms/step - loss: 0.2227 - accuracy:
0.9041
Epoch 13/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2200 - accuracy:
0.9051
Epoch 14/40
69/69 [==============================] - 3s 50ms/step - loss: 0.2162 - accuracy:
0.9064
Epoch 15/40
69/69 [==============================] - 3s 48ms/step - loss: 0.2143 - accuracy:
0.9069
Epoch 16/40
69/69 [==============================] - 3s 46ms/step - loss: 0.2115 - accuracy:
0.9076
Epoch 17/40
69/69 [==============================] - 3s 49ms/step - loss: 0.2107 - accuracy:
0.9080
Epoch 18/40
69/69 [==============================] - 3s 47ms/step - loss: 0.2081 - accuracy:
0.9085
Epoch 19/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2061 - accuracy:
0.9091
Epoch 20/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2041 - accuracy:
```

```
0.9098
Epoch 21/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2032 - accuracy:
0.9099
Epoch 22/40
69/69 [==============================] - 4s 51ms/step - loss: 0.2013 - accuracy:
0.9104
Epoch 23/40
69/69 [==============================] - 3s 49ms/step - loss: 0.2010 - accuracy:
0.9105
Epoch 24/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1994 - accuracy:
0.9108
Epoch 25/40
69/69 [==============================] - 3s 46ms/step - loss: 0.1986 - accuracy:
0.9109
Epoch 26/40
69/69 [==============================] - 3s 47ms/step - loss: 0.1972 - accuracy:
0.9112
Epoch 27/40
69/69 [==============================] - 4s 52ms/step - loss: 0.1954 - accuracy:
0.9114
Epoch 28/40
69/69 [==============================] - 4s 59ms/step - loss: 0.1948 - accuracy:
0.9119
Epoch 29/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1942 - accuracy:
0.9120
Epoch 30/40
69/69 [==============================] - 3s 48ms/step - loss: 0.1938 - accuracy:
0.9121
Epoch 31/40
69/69 [==============================] - 3s 50ms/step - loss: 0.1926 - accuracy:
0.9121
Epoch 32/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1920 - accuracy:
0.9121
Epoch 33/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1908 - accuracy:
0.9124
Epoch 34/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1907 - accuracy:
0.9124
Epoch 35/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1899 - accuracy:
0.9126
Epoch 36/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1892 - accuracy:
```

```
0.9125
Epoch 37/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1886 - accuracy:
0.9127
Epoch 38/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1881 - accuracy:
0.9128
Epoch 39/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1872 - accuracy:
0.9130
Epoch 40/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1873 - accuracy:
0.9130
2388/2388 [==============================] - 4s 2ms/step - loss: 0.2335 -
accuracy: 0.9127
Epoch 1/40
69/69 [==============================] - 5s 45ms/step - loss: 0.5239 - accuracy:
0.7822
Epoch 2/40
69/69 [==============================] - 3s 44ms/step - loss: 0.3658 - accuracy:
0.8642
Epoch 3/40
69/69 [==============================] - 3s 45ms/step - loss: 0.3179 - accuracy:
0.8742
Epoch 4/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2902 - accuracy:
0.8790
Epoch 5/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2717 - accuracy:
0.8835
Epoch 6/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2588 - accuracy:
0.8881
Epoch 7/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2483 - accuracy:
0.8920
Epoch 8/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2402 - accuracy:
0.8952
Epoch 9/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2344 - accuracy:
0.8979
Epoch 10/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2287 - accuracy:
0.8999
Epoch 11/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2252 - accuracy:
0.9016
```

```
Epoch 12/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2215 - accuracy:
0.9032
Epoch 13/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2174 - accuracy:
0.9044
Epoch 14/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2151 - accuracy:
0.9057
Epoch 15/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2125 - accuracy:
0.9065
Epoch 16/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2104 - accuracy:
0.9070
Epoch 17/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2090 - accuracy:
0.9077
Epoch 18/40
69/69 [==============================] - 3s 48ms/step - loss: 0.2065 - accuracy:
0.9084
Epoch 19/40
69/69 [==============================] - 4s 60ms/step - loss: 0.2048 - accuracy:
0.9089
Epoch 20/40
69/69 [==============================] - 4s 51ms/step - loss: 0.2035 - accuracy:
0.9094
Epoch 21/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2020 - accuracy:
0.9096
Epoch 22/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2007 - accuracy:
0.9101
Epoch 23/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2001 - accuracy:
0.9102
Epoch 24/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1986 - accuracy:
0.9105
Epoch 25/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1980 - accuracy:
0.9106
Epoch 26/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1967 - accuracy:
0.9110
Epoch 27/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1952 - accuracy:
0.9114
```

```
Epoch 28/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1953 - accuracy:
0.9113
Epoch 29/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1949 - accuracy:
0.9114
Epoch 30/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1937 - accuracy:
0.9118
Epoch 31/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1931 - accuracy:
0.9119
Epoch 32/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1922 - accuracy:
0.9120
Epoch 33/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1921 - accuracy:
0.9120
Epoch 34/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1911 - accuracy:
0.9123
Epoch 35/40
69/69 [==============================] - 3s 48ms/step - loss: 0.1906 - accuracy:
0.9124
Epoch 36/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1902 - accuracy:
0.9124
Epoch 37/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1900 - accuracy:
0.9126
Epoch 38/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1893 - accuracy:
0.9127
Epoch 39/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1889 - accuracy:
0.9127
Epoch 40/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1883 - accuracy:
0.9128
2388/2388 [==============================] - 3s 1ms/step - loss: 0.2339 -
accuracy: 0.9127
Epoch 1/40
69/69 [==============================] - 5s 46ms/step - loss: 0.5666 - accuracy:
0.7845
Epoch 2/40
69/69 [==============================] - 3s 42ms/step - loss: 0.3896 - accuracy:
0.8681
Epoch 3/40
```

```
69/69 [==============================] - 3s 40ms/step - loss: 0.3337 - accuracy:
0.8775
Epoch 4/40
69/69 [==============================] - 3s 41ms/step - loss: 0.3026 - accuracy:
0.8827
Epoch 5/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2833 - accuracy:
0.8866
Epoch 6/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2693 - accuracy:
0.8901
Epoch 7/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2586 - accuracy:
0.8932
Epoch 8/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2509 - accuracy:
0.8959
Epoch 9/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2443 - accuracy:
0.8983
Epoch 10/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2381 - accuracy:
0.9002
Epoch 11/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2348 - accuracy:
0.9016
Epoch 12/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2289 - accuracy:
0.9033
Epoch 13/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2258 - accuracy:
0.9042
Epoch 14/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2220 - accuracy:
0.9054
Epoch 15/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2189 - accuracy:
0.9063
Epoch 16/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2159 - accuracy:
0.9069
Epoch 17/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2144 - accuracy:
0.9073
Epoch 18/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2115 - accuracy:
0.9081
Epoch 19/40
```

```
69/69 [==============================] - 3s 42ms/step - loss: 0.2086 - accuracy:
0.9084
Epoch 20/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2074 - accuracy:
0.9089
Epoch 21/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2063 - accuracy:
0.9092
Epoch 22/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2045 - accuracy:
0.9096
Epoch 23/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2029 - accuracy:
0.9098
Epoch 24/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2014 - accuracy:
0.9100
Epoch 25/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2002 - accuracy:
0.9106
Epoch 26/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1993 - accuracy:
0.9107
Epoch 27/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1980 - accuracy:
0.9109
Epoch 28/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1971 - accuracy:
0.9110
Epoch 29/40
69/69 [==============================] - 4s 59ms/step - loss: 0.1970 - accuracy:
0.9112
Epoch 30/40
69/69 [==============================] - 4s 58ms/step - loss: 0.1953 - accuracy:
0.9114
Epoch 31/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1954 - accuracy:
0.9114
Epoch 32/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1938 - accuracy:
0.9117
Epoch 33/40
69/69 [==============================] - 4s 58ms/step - loss: 0.1933 - accuracy:
0.9116
Epoch 34/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1924 - accuracy:
0.9118
Epoch 35/40
```

```
69/69 [==============================] - 3s 43ms/step - loss: 0.1920 - accuracy:
0.9120
Epoch 36/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1916 - accuracy:
0.9119
Epoch 37/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1914 - accuracy:
0.9121
Epoch 38/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1905 - accuracy:
0.9122
Epoch 39/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1905 - accuracy:
0.9121
Epoch 40/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1900 - accuracy:
0.9121
2388/2388 [==============================] - 3s 1ms/step - loss: 0.2314 -
accuracy: 0.9146
Epoch 1/40
69/69 [==============================] - 4s 43ms/step - loss: 0.5470 - accuracy:
0.7795
Epoch 2/40
69/69 [==============================] - 3s 41ms/step - loss: 0.3785 - accuracy:
0.8660
Epoch 3/40
69/69 [==============================] - 3s 40ms/step - loss: 0.3256 - accuracy:
0.8761
Epoch 4/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2976 - accuracy:
0.8808
Epoch 5/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2791 - accuracy:
0.8848
Epoch 6/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2659 - accuracy:
0.8882
Epoch 7/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2550 - accuracy:
0.8915
Epoch 8/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2463 - accuracy:
0.8944
Epoch 9/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2398 - accuracy:
0.8969
Epoch 10/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2338 - accuracy:
```

```
0.8988
Epoch 11/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2284 - accuracy:
0.9007
Epoch 12/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2236 - accuracy:
0.9026
Epoch 13/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2202 - accuracy:
0.9038
Epoch 14/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2165 - accuracy:
0.9048
Epoch 15/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2143 - accuracy:
0.9057
Epoch 16/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2116 - accuracy:
0.9066
Epoch 17/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2089 - accuracy:
0.9076
Epoch 18/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2067 - accuracy:
0.9082
Epoch 19/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2051 - accuracy:
0.9087
Epoch 20/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2042 - accuracy:
0.9092
Epoch 21/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2020 - accuracy:
0.9099
Epoch 22/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2009 - accuracy:
0.9099
Epoch 23/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1990 - accuracy:
0.9105
Epoch 24/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1989 - accuracy:
0.9104
Epoch 25/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1972 - accuracy:
0.9108
Epoch 26/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1967 - accuracy:
```

```
0.9109
Epoch 27/40
69/69 [==============================] - 3s 50ms/step - loss: 0.1952 - accuracy:
0.9115
Epoch 28/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1941 - accuracy:
0.9118
Epoch 29/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1942 - accuracy:
0.9117
Epoch 30/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1929 - accuracy:
0.9119
Epoch 31/40
69/69 [==============================] - 3s 46ms/step - loss: 0.1924 - accuracy:
0.9122
Epoch 32/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1916 - accuracy:
0.9123
Epoch 33/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1907 - accuracy:
0.9124
Epoch 34/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1910 - accuracy:
0.9122
Epoch 35/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1904 - accuracy:
0.9124
Epoch 36/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1897 - accuracy:
0.9126
Epoch 37/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1893 - accuracy:
0.9125
Epoch 38/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1888 - accuracy:
0.9127
Epoch 39/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1886 - accuracy:
0.9127
Epoch 40/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1882 - accuracy:
0.9129
2388/2388 [==============================] - 3s 1ms/step - loss: 0.2329 -
accuracy: 0.9137
Epoch 1/40
69/69 [==============================] - 5s 44ms/step - loss: 0.5342 - accuracy:
0.7785
```

```
Epoch 2/40
69/69 [==============================] - 3s 43ms/step - loss: 0.3667 - accuracy:
0.8654
Epoch 3/40
69/69 [==============================] - 3s 42ms/step - loss: 0.3169 - accuracy:
0.8756
Epoch 4/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2902 - accuracy:
0.8812
Epoch 5/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2729 - accuracy:
0.8852
Epoch 6/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2607 - accuracy:
0.8889
Epoch 7/40
69/69 [==============================] - 3s 46ms/step - loss: 0.2509 - accuracy:
0.8918
Epoch 8/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2438 - accuracy:
0.8940
Epoch 9/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2373 - accuracy:
0.8968
Epoch 10/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2332 - accuracy:
0.8981
Epoch 11/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2292 - accuracy:
0.9001
Epoch 12/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2250 - accuracy:
0.9016
Epoch 13/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2218 - accuracy:
0.9029
Epoch 14/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2196 - accuracy:
0.9039
Epoch 15/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2159 - accuracy:
0.9051
Epoch 16/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2136 - accuracy:
0.9058
Epoch 17/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2115 - accuracy:
0.9067
```

```
Epoch 18/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2103 - accuracy:
0.9074
Epoch 19/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2072 - accuracy:
0.9083
Epoch 20/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2057 - accuracy:
0.9086
Epoch 21/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2052 - accuracy:
0.9089
Epoch 22/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2035 - accuracy:
0.9093
Epoch 23/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2020 - accuracy:
0.9097
Epoch 24/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2009 - accuracy:
0.9102
Epoch 25/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1995 - accuracy:
0.9105
Epoch 26/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1985 - accuracy:
0.9108
Epoch 27/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1978 - accuracy:
0.9110
Epoch 28/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1968 - accuracy:
0.9112
Epoch 29/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1964 - accuracy:
0.9114
Epoch 30/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1952 - accuracy:
0.9114
Epoch 31/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1939 - accuracy:
0.9117
Epoch 32/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1937 - accuracy:
0.9119
Epoch 33/40
69/69 [==============================] - 3s 47ms/step - loss: 0.1933 - accuracy:
0.9119
```

```
Epoch 34/40
69/69 [==============================] - 3s 45ms/step - loss: 0.1922 - accuracy:
0.9121
Epoch 35/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1919 - accuracy:
0.9121
Epoch 36/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1901 - accuracy:
0.9125
Epoch 37/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1904 - accuracy:
0.9124
Epoch 38/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1899 - accuracy:
0.9124
Epoch 39/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1886 - accuracy:
0.9126
Epoch 40/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1889 - accuracy:
0.9127
2388/2388 [==============================] - 3s 1ms/step - loss: 0.2327 -
accuracy: 0.9131
Epoch 1/40
69/69 [==============================] - 4s 43ms/step - loss: 0.5320 - accuracy:
0.7727
Epoch 2/40
69/69 [==============================] - 3s 44ms/step - loss: 0.3573 - accuracy:
0.8650
Epoch 3/40
69/69 [==============================] - 3s 43ms/step - loss: 0.3118 - accuracy:
0.8761
Epoch 4/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2878 - accuracy:
0.8822
Epoch 5/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2697 - accuracy:
0.8874
Epoch 6/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2583 - accuracy:
0.8911
Epoch 7/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2501 - accuracy:
0.8942
Epoch 8/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2423 - accuracy:
0.8967
Epoch 9/40
```

```
69/69 [==============================] - 3s 45ms/step - loss: 0.2361 - accuracy:
0.8991
Epoch 10/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2308 - accuracy:
0.9007
Epoch 11/40
69/69 [==============================] - 3s 40ms/step - loss: 0.2268 - accuracy:
0.9021
Epoch 12/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2238 - accuracy:
0.9035
Epoch 13/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2198 - accuracy:
0.9049
Epoch 14/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2174 - accuracy:
0.9055
Epoch 15/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2151 - accuracy:
0.9065
Epoch 16/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2129 - accuracy:
0.9068
Epoch 17/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2111 - accuracy:
0.9076
Epoch 18/40
69/69 [==============================] - 3s 42ms/step - loss: 0.2084 - accuracy:
0.9082
Epoch 19/40
69/69 [==============================] - 3s 47ms/step - loss: 0.2073 - accuracy:
0.9087
Epoch 20/40
69/69 [==============================] - 3s 45ms/step - loss: 0.2048 - accuracy:
0.9092
Epoch 21/40
69/69 [==============================] - 3s 43ms/step - loss: 0.2041 - accuracy:
0.9094
Epoch 22/40
69/69 [==============================] - 3s 44ms/step - loss: 0.2027 - accuracy:
0.9097
Epoch 23/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2008 - accuracy:
0.9102
Epoch 24/40
69/69 [==============================] - 3s 41ms/step - loss: 0.2004 - accuracy:
0.9105
Epoch 25/40
```

```
69/69 [==============================] - 3s 42ms/step - loss: 0.1989 - accuracy:
0.9107
Epoch 26/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1987 - accuracy:
0.9108
Epoch 27/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1970 - accuracy:
0.9113
Epoch 28/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1958 - accuracy:
0.9114
Epoch 29/40
69/69 [==============================] - 3s 44ms/step - loss: 0.1952 - accuracy:
0.9114
Epoch 30/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1948 - accuracy:
0.9115
Epoch 31/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1937 - accuracy:
0.9118
Epoch 32/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1930 - accuracy:
0.9118
Epoch 33/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1922 - accuracy:
0.9120
Epoch 34/40
69/69 [==============================] - 3s 41ms/step - loss: 0.1923 - accuracy:
0.9118
Epoch 35/40
69/69 [==============================] - 3s 40ms/step - loss: 0.1912 - accuracy:
0.9122
Epoch 36/40
69/69 [==============================] - 3s 42ms/step - loss: 0.1906 - accuracy:
0.9123
Epoch 37/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1902 - accuracy:
0.9122
Epoch 38/40
69/69 [==============================] - 3s 43ms/step - loss: 0.1891 - accuracy:
0.9126
Epoch 39/40
69/69 [==============================] - 3s 48ms/step - loss: 0.1889 - accuracy:
0.9125
Epoch 40/40
69/69 [==============================] - 4s 51ms/step - loss: 0.1881 - accuracy:
0.9126
2388/2388 [==============================] - 4s 1ms/step - loss: 0.2331 -
```

```
accuracy: 0.9127
The mean is: 0.9131726324558258
The sd is: 0.001106874401581888
```

Answer:

- **Question 19**: From the output the mean of the test accuracy is around 0.9132 and the standard deviation is around 0.0011.
- **Question 20**: The main advantage of dropout compared to CV is that we only need to train one model, this makes it faster to estimate the test uncertainty. If we had a model that took 24 hours to train, it takes a bit more than 24 hours to estimate the test uncertainty with dropout. For CV it would take more than 10 days.

23 Part 22: DNN regression

A similar DNN can be used for regression, instead of classification.

Question 21: How would you change the DNN used in this lab in order to use it for regression instead?

Answer:

- **Question 21**: We would change the metrics in model.compile to MSE. We would also change the activation function in the final layer to none. This will give is an output that can take on any real number.

23.1 Report

Send in this jupyter notebook, with answers to all questions.