OTTO Classification Competition Assignment

Make a copy for yourself

Click "File" -> Save a Copy in Drive on Colab editor.

Team members:

Name: Patrick Pfenning

Submission

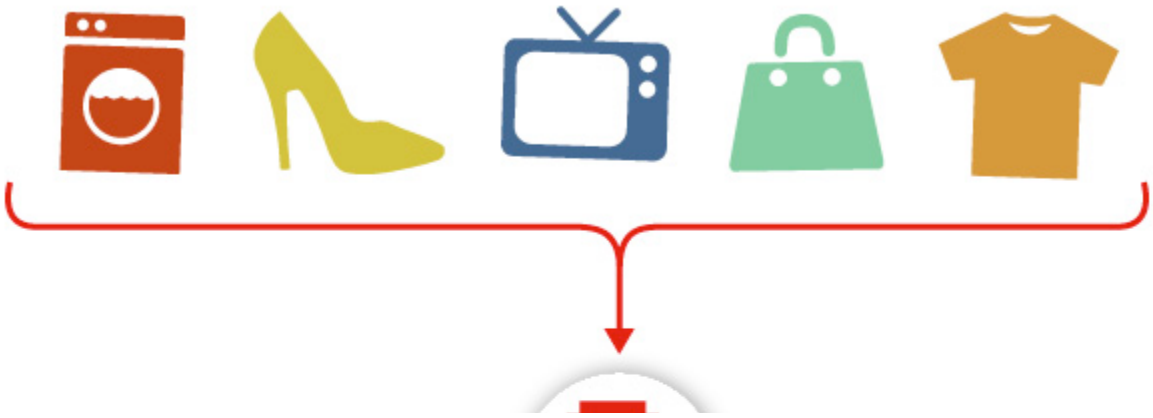
You need to subit your code to Kaggle to get a score and then upload your results to Brightspace.

Only one submission per team is needed.

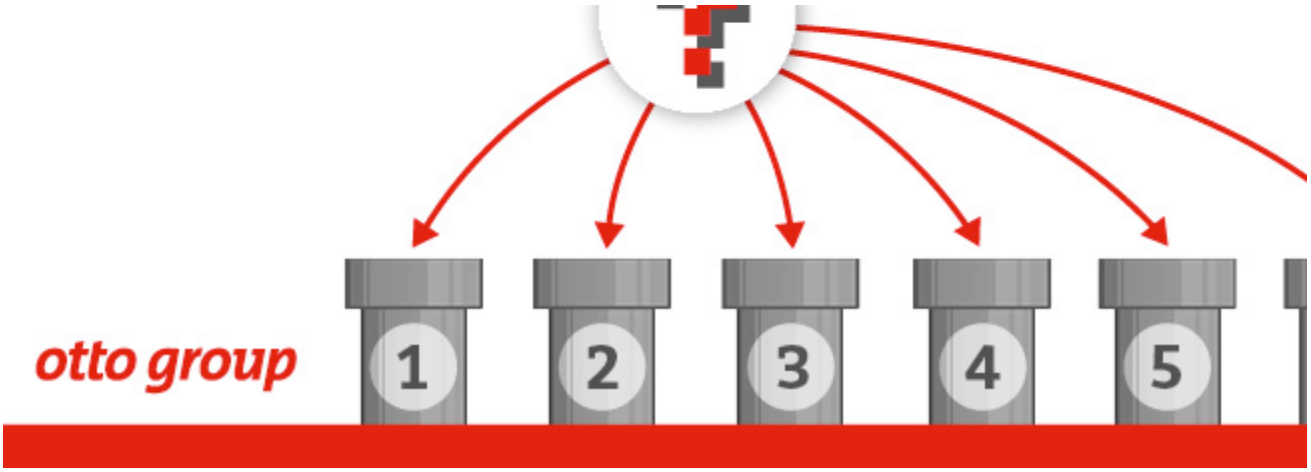
About the Company

The Otto Group is one of the world's biggest e-commerce companies, with subsidiaries in more than 20 countries, including Crate & Barrel (USA), Otto.de (Germany) and 3 Suisses (France). We are selling millions of products worldwide every day, with several thousand products being added to our product line.

A consistent analysis of the performance of our products is crucial. However, due to our diverse global infrastructure, many identical products get classified differently. Therefore, the quality of our product analysis depends heavily on the ability to accurately cluster similar products. The better the classification, the more insights we can generate about our product range.



✓ 0s completed at 11:47 PM



From Kaggle: https://www.kaggle.com/c/otto-group-product-classification-challenge/data?select=train.csv

Data

Each row corresponds to a single product. There are a total of 93 numerical features, which represent counts of different events. All features have been obfuscated and will not be defined any further.

There are nine categories for all products. Each target category represents one of our most important product categories (like fashion, electronics, etc.). The products for the training and testing sets are selected randomly.

Competition

Held in 2015 for $10,000 prize money. We can still submit!

Example solution:

https://www.kaggle.com/sachinsharma1123/otto-group-classification-acc-82

Download Data

We made the competition files avaiable to you.

```
#Files made public in class G Drive:
#get their link as viewer: https://drive.google.com/file/d/1-JlwRnCsQ17F3LCxlYg
!gdown --id 1-JlwRnCsQ17F3LCxlYg1u5b8GKA12Yc5 #train.csv
!gdown --id 1-DGMzegsuf9q1RVQID6SUfaHWoxmO1D5 #test.csv
!gdown --id 1-DhQHpxPknQStR_Y-VTQHnU7C2jlwBSD #sample_submission.csv
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Optio
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1-JlwRnCsQ17F3LCxlYg1u5b8GKA12Yc5
To: /content/train.csv
100% 12.4M/12.4M [00:00<00:00, 227MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Optio
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1-DGMzegsuf9q1RVQID6SUfaHWoxmO1D5
To: /content/test.csv
100% 27.9M/27.9M [00:00<00:00, 74.6MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Optio
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1-DhQHpxPknQStR_Y-VTQHnU7C2jlwBSD
To: /content/sampleSubmission.csv
100% 3.50M/3.50M [00:00<00:00, 190MB/s]
```

```
 1 import pandas as pd
 2 import numpy as np
 3 from sklearn import preprocessing
 4 from matplotlib import pyplot as plt
 5
 6
 7 train_org = pd.read_csv("train.csv")
 8 test_org = pd.read_csv("test.csv")
 9 sample_org = pd.read_csv("sampleSubmission.csv")
10
11 train_org[:10]
```

	id	feat_1	feat_2	feat_3	feat_4	feat_5	feat_6	feat_7	feat_8	feat_9	.
0	1	1	0	0	0	0	0	0	0	0	
1	2	0	0	0	0	0	0	0	1	0	
2	3	0	0	0	0	0	0	0	1	0	
3	4	1	0	0	1	6	1	5	0	0	
4	5	0	0	0	0	0	0	0	0	0	
5	6	2	1	0	0	7	0	0	0	0	
6	7	2	0	0	0	0	0	0	2	0	
7	8	0	0	0	0	0	0	0	0	0	
8	9	0	0	0	0	0	0	0	4	0	
9	10	0	0	0	0	0	0	1	0	0	

10 rows × 95 columns

```
1 train_labels = train_org.target.values
2 train_labels[:-10]
```

```
array(['Class_1', 'Class_1', 'Class_1', ..., 'Class_9', 'Class_9',
       'Class_9'], dtype=object)
```

```
1 np.unique(train_labels)
```

```
array(['Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5', 'Class_6',
       'Class_7', 'Class_8', 'Class_9'], dtype=object)
```

```
1 #drop ids and labels
2 train = train_org.drop('id',axis = 1)
3 train = train.drop('target',axis = 1)
4 test = test_org.drop('id',axis = 1)
5
6 # convert labels to numeric values
7 lbl_enc = preprocessing.LabelEncoder()
8 labels = lbl_enc.fit_transform(train_labels)
9 np.unique(labels)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
1 train = train.astype("float32")
2 test = test.astype("float32")
3
4 print(train.shape, test.shape, train_labels.shape)
```

```
(61878, 93) (144368, 93) (61878,)
```

```
1 train.describe()
```

	feat_1	feat_2	feat_3	feat_4	feat_5	feat_
count	61878.00000	61878.000000	61878.000000	61878.000000	61878.000000	61878.0000(
mean	0.38668	0.263066	0.901467	0.779081	0.071043	0.02569
std	1.52533	1.252073	2.934818	2.788005	0.438902	0.21533
min	0.00000	0.000000	0.000000	0.000000	0.000000	0.0000(
25%	0.00000	0.000000	0.000000	0.000000	0.000000	0.0000(
50%	0.00000	0.000000	0.000000	0.000000	0.000000	0.0000(
75%	0.00000	0.000000	0.000000	0.000000	0.000000	0.0000(
max	61.00000	51.000000	64.000000	70.000000	19.000000	10.0000(

8 rows × 93 columns

```
1 np.unique(labels)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

Homework

Copy and paste network from Workshop 2 below.

Update the output layer accordingly.

The network architecture

```
 1 import tensorflow as tf
 2 from dataclasses import dataclass
 3 from tensorflow.keras import datasets, layers, models
 4 from keras.callbacks import CSVLogger
 5 import matplotlib.pyplot as plt
 6 import seaborn as sns
 7 import pandas as pd
 8 import os
 9 import pickle as pkl
10 sns.set()
11
12 @dataclass
13 class myNN:
14
15     name: str
16     X_train: pd.DataFrame = train
17     y_train: np.ndarray = labels
18     X_test: pd.DataFrame = test
19     max_layers: int = 10
20     epochs: int = 10
21     refresh: bool = False
22
23     def __post_init__(self, refresh=False):
24         self.model, self.history = self.get_model()
25         self.show()
26
27     def get_model(self):
28         fname = 'comp_model'
```

```
        if (os.path.exists(f'{fname}.sav') and os.path.exists(f'{fname}.log'))
            model = models.load_model(f'{fname}.sav')
        else:
            model = self.__get_model(fname)
            model.save(f'{fname}.sav')
        return model, pd.read_csv(f'{fname}.log', sep=',', engine='python')

    def __get_model(self, fname):
        model = models.Sequential(name=self.name)
        class_cnt = len(np.unique(self.y_train))
        model.add(layers.Dense(input_dim=self.X_test.shape[1], units=np.power(2
        for n in np.arange(self.max_layers)[::-1]:
            nodes = np.power(2, n)
            if nodes > class_cnt:
                model.add(layers.Dense(units=nodes, name=f'hidden{n}', activati
        model.add(layers.Dense(units=class_cnt, name='output', activation='soft
        model.compile(
            optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(),
            metrics=['accuracy', 'mse']
        )
        csv_logger = CSVLogger(f'{fname}.log', separator=',', append=False)
        model.fit(
            self.X_train,
            self.y_train,
            epochs=self.epochs,
            callbacks=[csv_logger]
        )
        return model

    def show(self):
        cycle = plt.rcParams['axes.prop_cycle'].by_key()['color']
        lines = ["-","--","-.",":"]
        fig, axs = plt.subplots(nrows=3, ncols=1, sharex=True)
        plt.tight_layout()
        metrics = ['Accuracy', 'Loss', 'MSE']
        for n, metric in enumerate(metrics):
            index = n + np.int((n)/len(metrics))
            axs[n].plot(
                self.history[metric.lower()],
                linewidth=2.0+index,
                color=cycle[n],
                linestyle = lines[index],
                label=f'Train {metric}'
            )
            axs[n].set_title(f"Model {metric}")
        plt.xlabel('Epochs', fontsize=20)
        plt.savefig(f"comp_model.png")

model1 = myNN(name='otto', refresh=True, max_layers=11, epochs=50)
```

```
Epoch 1/50
1934/1934 [==============================] - 61s 31ms/step - loss: 0.7212 - a
Epoch 2/50
1934/1934 [==============================] - 62s 32ms/step - loss: 0.5789 - a
Epoch 3/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.5238 - a
Epoch 4/50
1934/1934 [==============================] - 61s 32ms/step - loss: 0.4744 - a
Epoch 5/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.4382 - a
Epoch 6/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.4074 - a
Epoch 7/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.3716 - a
Epoch 8/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.3440 - a
Epoch 9/50
1934/1934 [==============================] - 61s 32ms/step - loss: 0.3151 - a
Epoch 10/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.2922 - a
Epoch 11/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.2748 - a
Epoch 12/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.2469 - a
Epoch 13/50
1934/1934 [==============================] - 61s 32ms/step - loss: 0.2327 - a
Epoch 14/50
1934/1934 [==============================] - 61s 31ms/step - loss: 0.2128 - a
Epoch 15/50
1934/1934 [==============================] - 61s 31ms/step - loss: 0.1988 - a
Epoch 16/50
1934/1934 [==============================] - 61s 31ms/step - loss: 0.1865 - a
Epoch 17/50
1934/1934 [==============================] - 61s 32ms/step - loss: 0.1671 - a
Epoch 18/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1745 - a
Epoch 19/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1624 - a
Epoch 20/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1643 - a
Epoch 21/50
1934/1934 [==============================] - 61s 32ms/step - loss: 0.1349 - a
Epoch 22/50
1934/1934 [==============================] - 61s 31ms/step - loss: 0.1342 - a
Epoch 23/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1287 - a
Epoch 24/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1289 - a
Epoch 25/50
1934/1934 [==============================] - 61s 32ms/step - loss: 0.1518 - a
Epoch 26/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1087 - a
```

```
Epoch 27/50
1934/1934 [==============================] - 60s 31ms/step - loss: 0.1164 - a
```

Add at least 1 hidden layer.

We can add more hidden Dense layers with varying number of neurons.

The "output layer" is problem specific. How many classes do we have to predict?

The compilation step

```
1 # model.compile(?)
```

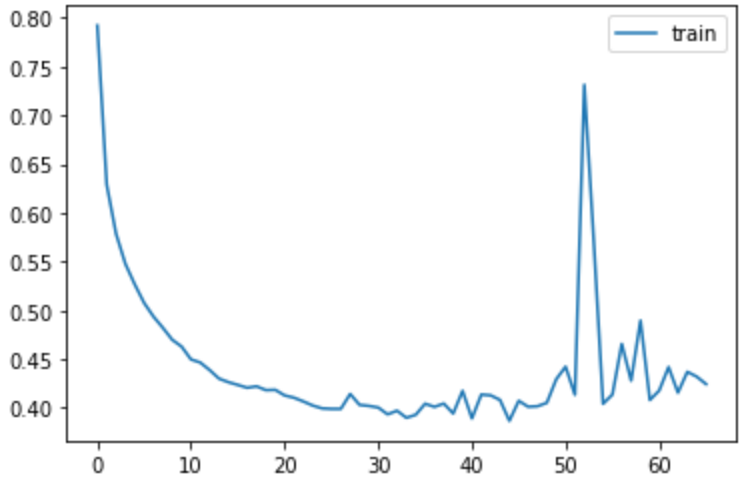
```
1 # history = model.fit(?)
```

```
Epoch 1/66
484/484 [==============================] - 5s 9ms/step - loss: 0.7923 - accur
Epoch 2/66
484/484 [==============================] - 4s 9ms/step - loss: 0.6292 - accur
Epoch 3/66
484/484 [==============================] - 4s 9ms/step - loss: 0.5787 - accur
Epoch 4/66
484/484 [==============================] - 4s 9ms/step - loss: 0.5477 - accur
Epoch 5/66
484/484 [==============================] - 4s 9ms/step - loss: 0.5267 - accur
Epoch 6/66
484/484 [==============================] - 4s 9ms/step - loss: 0.5077 - accur
Epoch 7/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4937 - accur
Epoch 8/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4822 - accur
Epoch 9/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4700 - accur
Epoch 10/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4630 - accur
Epoch 11/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4498 - accur
Epoch 12/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4466 - accur
Epoch 13/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4388 - accur
Epoch 14/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4301 - accur
Epoch 15/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4265 - accur
Epoch 16/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4237 - accur
Epoch 17/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4207 - accur
Epoch 18/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4222 - accur
```

```
Epoch 19/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4183 - accur
Epoch 20/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4186 - accur
Epoch 21/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4129 - accur
Epoch 22/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4105 - accur
Epoch 23/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4067 - accur
Epoch 24/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4025 - accur
Epoch 25/66
484/484 [==============================] - 4s 9ms/step - loss: 0.3996 - accur
Epoch 26/66
484/484 [==============================] - 4s 9ms/step - loss: 0.3991 - accur
Epoch 27/66
484/484 [==============================] - 4s 9ms/step - loss: 0.3990 - accur
Epoch 28/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4144 - accur
Epoch 29/66
484/484 [==============================] - 4s 9ms/step - loss: 0.4032 - accur
```

```
1 plt.plot(history.history['loss'], label='train')
2 plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f18e9fcfc50>
```



Submission

You need to subit your code to Kaggle to get a score and then upload your results to Brightspace.

Kaggle

Create a submission for Kaggle

The code below is to help you format your estimates to a submission friendly file. You shouldn't have to edit it.

Make sure that:

- your NN is named 'model'
- your test data is named 'test_sub'

```
#submissions: rename variables if needed
test_sub = test
model = model1.model

#Run model on test data
predictions = model.predict(test_sub)
predictions[0]
```

```
array([6.1803763e-07, 9.9977893e-01, 7.4204778e-05, 1.4361134e-04,
       2.8707069e-07, 6.3061452e-07, 1.2727301e-06, 4.1250198e-07,
       1.6369653e-07], dtype=float32)
```

```
predictions[0].argmax()
```

```
1
```

```
sub = pd.DataFrame(np.int32(predictions.round()))
sub
```

	0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	1	0	0	0
3	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...
144363	0	0	0	0	0	1	0	0	0
144364	0	0	1	0	0	0	0	0	0
144365	0	1	0	0	0	0	0	0	0
144366	0	0	0	1	0	0	0	0	0
144367	0	1	0	0	0	0	0	0	0

144368 rows × 9 columns

```
1 sub.columns = ['Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5', 'Class_6'
2 sub.insert(loc=0,column='id',value = test_org.id)
3 sub
```

	id	Class_1	Class_2	Class_3	Class_4	Class_5	Class_6	Class_7	C
0	1	0	1	0	0	0	0	0	
1	2	0	0	0	0	0	0	0	
2	3	0	0	0	0	0	1	0	
3	4	0	1	0	0	0	0	0	
4	5	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...
144363	144364	0	0	0	0	0	1	0	
144364	144365	0	0	1	0	0	0	0	
144365	144366	0	1	0	0	0	0	0	
144366	144367	0	0	0	1	0	0	0	
144367	144368	0	1	0	0	0	0	0	

144368 rows × 10 columns

```
1 sub.to_csv("kaggle_otto_submission.csv",index=False)
```

Right click on this file and download it to your machine.

Go to: https://www.kaggle.com/c/otto-group-product-classification-challenge/leaderboard#score

Create an account and click "Late Submission" to score your work!

Scoring:

Kaggle will run your submission and rank your result.

You want a score less than 10; the closer to 0, the better!

```
1
```

Brightspace (Graded)

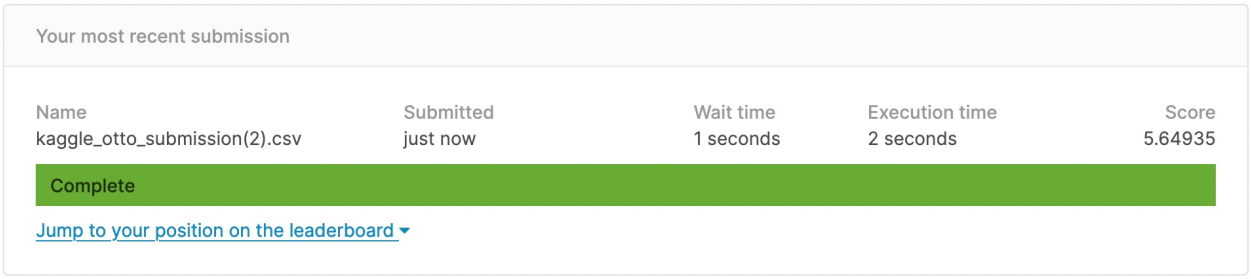
In Colab window:

in Colab window:

- Download your notebook: Click "File" -> Download - > Download .ipynb
- Print your notebook: Click "File" -> Print and then save a copy as PDF

Go to BrightSpace, upload:

- Notebook as .ipynb file,
- print out as a PDF,
- Screenshot of your score from Kaggle. Example:



1