ML Prediction - With Neural Networks

Help Functions

This function is meant for combining all data we have into a single Dataframe which contains a row for each product, thus making data like the nutrients available at the product level.

We also merge it with our previous predictions from the CNN we created, and the predictions are added as another feature.

In [361]:

```
def prepare_unified_df(test=False):
    ntr = pd.read_csv("./data/nutrients.csv")
    food_ntr = pd.read_csv("./data/food_nutrients.csv")

    ntr_combined = food_ntr.merge(ntr, on='nutrient_id', how='left')
    all_ntr_lists = ntr_combined.groupby("idx")["name"].apply(list).apply(lambda x: " ".join(x))
    idx_nutrients = pd.DataFrame(all_ntr_lists).rename(columns={"name": "nutrients_list"})

    if test:
        food = pd.read_csv("./data/food_test.csv")
    else:
        food = pd.read_csv("./data/food_train.csv")

    comb = food.merge(idx_nutrients, on="idx")

    return comb
```

In [362]:

```
def load_train_data():
    # Returns train data seperated in to features (X) df and target series (y)
    df = prepare_unified_df(test=False)
    # Shuffle the dataset
#     df = df.sample(frac=1, random_state=1).reset_index(drop=True)
    X = df.drop(["category"], axis=1)
    y = df["category"]
    return X, y
```

In [363]:

```
def load_test_data():
    # Returns features (X) of test data
    return prepare_unified_df(test=True)
```

This function is meant for:

- "Cleaning" existing columns like 'description', 'ingredients' and 'household_serving_fulltext' by filling Null values and more
- Combining the brand's name, the descrition and the household_serving_fulltext together to 1 column named 'text' - just becuase they are pretty similar in their meaning.

In [364]:

```
def preprocess(df, with_cnn=False):
    df["description"] = df["description"].str.strip('“').str.strip('”').str.strip('"')
    df["brand"] = df["brand"].fillna("")
    df["ingredients"] = df["ingredients"].fillna("")  # Some Null values
    df["household_serving_fulltext"] = df["household_serving_fulltext"].fillna("")  # Some Null values
    df["text"] = reduce(lambda x, y: x + " " + y, (df[x].fillna("") for x in [
                                                                              "brand",
                                                                              "description",
                                                                              "household_serving_fulltext"
                                                                              ]))
    if with_cnn:
        df = df[["brand", "text", "ingredients", "nutrients_list", "serving_size", "cnn_prediction"]]
        return df

    df = df[["brand", "text", "ingredients", "nutrients_list", "serving_size"]]
    return df
```

Prepare input data

In [365]:

```
def add_num_of_ings_col(df):
    df['n_ingredients'] = df['ingredients'].apply(lambda x: len(str(x).split(',')))
    return df
```

In [366]:

```
def is_choc_in_row(v):
    for i in v:
        if type(i) == str:
            if 'chocolate' in i.lower().strip():
                return 'chocolate'
    return 'no chocolate'
```

In [367]:

```
def on_field(f: str, *vec) -> Pipeline:
    return make_pipeline(FunctionTransformer(itemgetter(f), validate=False), *vec)
```

In [368]:

```
def create_vectorizer(with_cnn=False):
    if with_cnn:
        vectorizer = make_union(
        on_field('brand', Tfidf(max_features=4000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('cnn_prediction', Tfidf(max_features=6, token_pattern='\w+', ngram_range=(1, 1))),
        on_field('text', Tfidf(max_features=10000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('ingredients', Tfidf(max_features=2000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('nutrients_list', Tfidf(max_features=2000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('is_chocolate_in_text', Tfidf(max_features=2, token_pattern='\w+', ngram_range=(1, 1))),
        on_field(['serving_size', 'n_ingredients'], FunctionTransformer(lambda x: x, validate=False)),
        n_jobs=4
        )
        return vectorizer

    vectorizer = make_union(
        on_field('brand', Tfidf(max_features=4000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('text', Tfidf(max_features=10000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('ingredients', Tfidf(max_features=2000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('nutrients_list', Tfidf(max_features=2000, token_pattern='\w+', ngram_range=(1, 2))),
        on_field('is_chocolate_in_text', Tfidf(max_features=2, token_pattern='\w+', ngram_range=(1, 1))),
        on_field(['serving_size', 'n_ingredients'], FunctionTransformer(lambda x: x, validate=False)),
        n_jobs=4
    )
    return vectorizer
```

This function is meant for preparing the X_train and X_test dataframes to be exactly as the NN model expects them to be. Here we:

- Preprocess the data
- Create the 'n_ingredients' 'is_chocolate_in_text' and features
- scale the 'serving_size' and 'n_ingredients' features
- Vectorize all text columns with TfidfVectorizer

(Code is hidden as function is unfortunately long, appears in notebook)

Prepare target data

This function is meant for preparing the y_train and y_test arrays to be exactly as the NN model expects them to be. Here we encode the labels, and then use OH encoding to transform the data into the correct format for the NN.

In [370]:

```
def prepare_targets(y_train, y_test, y_valid=None, has_valid=True):
    le = LabelEncoder()
    le.fit(y_train)

    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print(le_name_mapping)

    y_train_enc = le.transform(y_train)
    y_test_enc = le.transform(y_test)

    # One Hot encoding for output
    y_train_enc = np_utils.to_categorical(y_train_enc)
    y_test_enc = np_utils.to_categorical(y_test_enc)

    if has_valid:
        y_valid_enc = le.transform(y_valid)
        y_valid_enc = np_utils.to_categorical(y_valid_enc)

        return y_train_enc, y_valid_enc, y_test_enc

    return y_train_enc, y_test_enc
```

function for preparing all needed data for model: (Code is hidden, appears in notebook)

In [386]:

```
mapping = {
    0: "cakes_cupcakes_snack_cakes",
    1: "candy",
    2: "chips_pretzels_snacks",
    3: "chocolate",
    4: "cookies_biscuits",
    5: "popcorn_peanuts_seeds_related_snacks"
}
```

Function for plotting the train and validation accuracy:

In [372]:

```
def plot_accuracy(history):
    history_dict = history.history

    acc = history_dict['accuracy']
    val_acc = history_dict['val_accuracy']

    # range of X (no. of epochs)
    epochs = range(1, len(acc) + 1)

    # plot
    plt.plot(epochs, acc, 'r', label='Training accuracy')
    plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

First model creation:

This model configuration was chosen after we tried several different parameters manually, such as number of layers (3 always seemed to work better than 2), number of nuerons per layer and the dropout rate. But we'll do some CV later!

In [373]:

```
def create_model(dp=0.4, optimizer='adam', num_nuerons=[128, 32, 16]):
    model_in = ks.Input(shape=(INPUT_SHAPE,), dtype='float32', sparse=True)
    out = ks.layers.Dense(num_nuerons[0], activation='relu')(model_in)
    out = ks.layers.Dropout(dp)(out)
    out = ks.layers.Dense(num_nuerons[1], activation='relu')(out)
    out = ks.layers.Dropout(dp)(out)
    out = ks.layers.Dense(num_nuerons[2], activation='relu')(out)
    out = ks.layers.Dense(6, activation='softmax')(out)
    model = ks.Model(model_in, out)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy']) # rmsprop/adam
    return model
```

In [374]:

```
# Early stopping callback
es = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    mode='min',
    patience=10,  # stop training when there's no improvement in val_loss for 10 consecutive epochs
    restore_best_weights=True
)
```

1st Model

Get Data & Run NN Model

In [345]:

```
# Get data
X_train, X_valid, X_test, y_train, y_valid, y_test = get_data_for_model(
    test_size=0.15,
    with_valid=True,
    valid_size=0.1
)
```

```
{'cakes_cupcakes_snack_cakes': 0, 'candy': 1, 'chips_pretzels_snacks': 2, 'chocolate': 3, 'cookies_b
iscuits': 4, 'popcorn_peanuts_seeds_related_snacks': 5}
Train:
        X_train: (24289, 16439) of float32
        y_train: (24289, 6) of float32
Validation:
        X_valid: (2699, 16439) of float32
        y_valid: (2699, 6) of float32
Test:
        X_test: (4763, 16439) of float32
        y_test: (4763, 6) of float32
```

In [ ]:

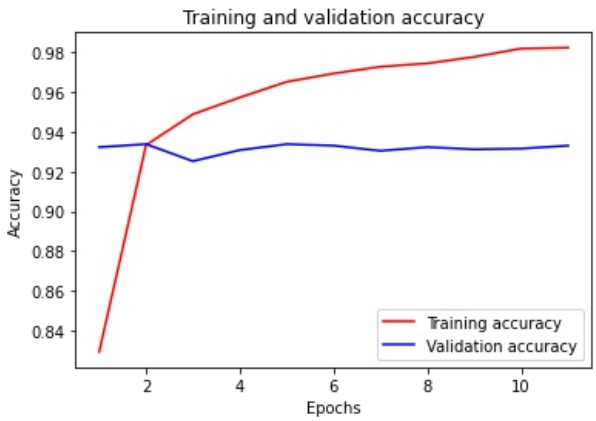
```
# Create model
INPUT_SHAPE = X_train.shape[1]
model = create_model()

# Fit model on training data
history = model.fit(
    X_train,
    y_train,
    callbacks=[es],
    epochs=10000,
    shuffle=True,
    validation_data=(X_valid, y_valid),
    verbose=2
)
```

Evaluate Model

Plotting training and validation accuracy:

In [348]:

```
plot_accuracy(history)
```



Classification Report on validation data:

(validation data was used by the model for estimating performance, but model was not trained on it!)

In [349]:

```
preds = model.predict(X_valid, verbose=0)
# Classification report
print(classification_report(y_valid.argmax(axis=1), preds.argmax(axis=1)))
_, accuracy = model.evaluate(X_valid, y_valid, verbose=0)
print('Accuracy on validation: %.2f' % (accuracy*100))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       335
           1       0.94      0.90      0.92       620
           2       0.98      0.96      0.97       335
           3       0.80      0.89      0.84       319
           4       0.96      0.93      0.94       454
           5       0.96      0.94      0.95       636

    accuracy                           0.93      2699
   macro avg       0.93      0.93      0.93      2699
weighted avg       0.93      0.93      0.93      2699

Accuracy on validation: 93.22
```

Classification Report on test data:

In [350]:

```
preds = model.predict(X_test, verbose=0)
# Classification report
print(classification_report(y_test.argmax(axis=1), preds.argmax(axis=1)))

_, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test: %.2f' % (accuracy*100))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       565
           1       0.96      0.91      0.93      1120
           2       0.97      0.96      0.97       536
           3       0.82      0.91      0.86       591
           4       0.93      0.95      0.94       818
           5       0.97      0.96      0.96      1133

    accuracy                           0.94      4763
   macro avg       0.93      0.94      0.94      4763
weighted avg       0.94      0.94      0.94      4763

Accuracy on test: 93.83
```

Before we continue, let's train this model on all data and save a prediction in the test set, in case this will be one of our final models:

In [351]:

```
train_data, y = load_train_data()
train_data, _, test_data = prepare_inputs(train_data, train_data, has_valid=False)

y, _ = prepare_targets(y, y, has_valid=False)

print("Train:")
print(f'\ttrain_data: {train_data.shape} of {train_data.dtype}')
print(f'\ty: {y.shape} of {y.dtype}')
print("Test:")
print(f'\ttest_data: {test_data.shape} of {test_data.dtype}')
```

```
{'cakes_cupcakes_snack_cakes': 0, 'candy': 1, 'chips_pretzels_snacks': 2, 'chocolate': 3, 'cookies_b
iscuits': 4, 'popcorn_peanuts_seeds_related_snacks': 5}
Train:
        train_data: (31751, 16446) of float32
        y: (31751, 6) of float32
Test:
        test_data: (3525, 16446) of float32
```

In [352]:

```
# Create model
INPUT_SHAPE = train_data.shape[1]
model = create_model()

# Fit model on all train data
history = model.fit(
    train_data,
    y,
    epochs=10,  # We see that there's almost no improvement after a couple of epochs, so we need it to avoid over
fitting.
    shuffle=True,
    verbose=0
)
```

Sanity check on train data

In [353]:

```
preds = model.predict(train_data, verbose=0)
# Classification report
print(classification_report(y.argmax(axis=1), preds.argmax(axis=1)))
_, accuracy = model.evaluate(train_data, y, verbose=0)
print('Accuracy on validation: %.2f' % (accuracy*100))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      3786
           1       0.99      0.97      0.98      7584
           2       1.00      1.00      1.00      3680
           3       0.94      0.98      0.96      3772
           4       1.00      0.99      0.99      5284
           5       1.00      0.99      1.00      7645

    accuracy                           0.99     31751
   macro avg       0.99      0.99      0.99     31751
weighted avg       0.99      0.99      0.99     31751

Accuracy on validation: 98.88
```

Predict on test data and save results

In [354]:

```
preds = model.predict(test_data).argmax(axis=1)

mapping = {
    0: "cakes_cupcakes_snack_cakes",
    1: "candy",
    2: "chips_pretzels_snacks",
    3: "chocolate",
    4: "cookies_biscuits",
    5: "popcorn_peanuts_seeds_related_snacks"
}

preds = [mapping[i] for i in preds]

test_indexes = load_test_data().idx
```

```
111/111 [==============================] - 0s 2ms/step
```

In [355]:

```
results_model_03 = pd.DataFrame({"idx": indexes, "pred_cat": preds})
results_model_03.to_csv("model_03.csv", index=False)

# For sanity check
t = load_test_data()
t["pred_cat"] = preds
t.to_csv("model_03_check.csv", index=False)
```

Notice: Here you see creation of model 3, which includes the "is_chocolate" feature. Model 1 which we created is the same model, but without this feature, and was run before this feature was added to our code.

Conclusions until now:

1) We see that the precision and recall of all 6 categories are pretty similar, so we can assure ourselves that using accuracy is indeed OK.

2) We see that we get a pretty good accuracy without needing to do any complicated manipulations on the tabular data, and overall it really makes sense, as we saw in the data exploration step that the data is mostly text, and that these texts, such as brand, description and ingredients, hold most of the information we need to know the category. So, by using Tfidf to convert the text to TF-IDF features, we really use all this information.

3) We also see that the validation accuracy doesn't really go up, so we need to be cautious of overfitting. It seems that increasing the epochs is definitely not the solution (we're just staying with the same validation accuracy), so improving the model with have to come from improving the hyperparameters or adding additional data.

4) We can see that the f1 score of category #3 (it's choclate) is very low compared to all other categories. That means that we do a not-so-good job at predicting the Choclate category, and we need to go back to the data and see if there's some information that we can use to improve this.

So, this is going to be our next step!

2nd Model: Adding additional params to improve *Choclate* classification

Updating relevant functions:

For now we create new versions of create_vectorizer and prepare_inputs, as we're not sure yet we want this feature in our final model (Code is hidden, appears in notebook)

Prepare data & Run NN model with new feature

In [381]:

```
# Prepare data
X, y = load_train_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=1)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.1, random_state=1)
# Use prepare_inputs_2 functions and not original one
X_train, X_valid, X_test = prepare_inputs_2(X_train, X_test, X_valid)
y_train, y_valid, y_test = prepare_targets(y_train, y_test, y_valid)
```

```
{'cakes_cupcakes_snack_cakes': 0, 'candy': 1, 'chips_pretzels_snacks': 2, 'chocolate': 3, 'cookies_b
iscuits': 4, 'popcorn_peanuts_seeds_related_snacks': 5}
```

In [ ]:

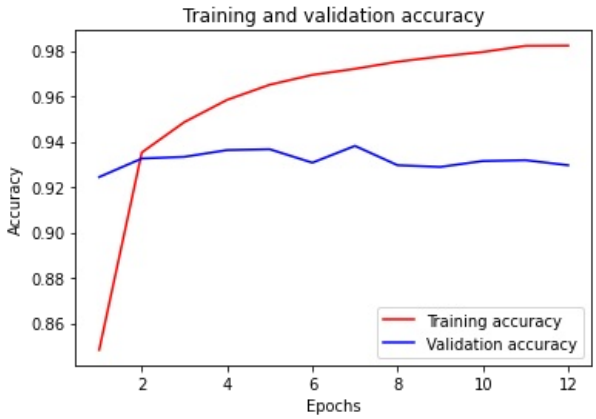
```
# Create model
INPUT_SHAPE = X_train.shape[1]
model = create_model()

# Fit model on training data
history = model.fit(
    X_train,
    y_train,
    callbacks=[es],
    epochs=10000,
    batch_size=16,
    shuffle=True,
    validation_data=(X_valid, y_valid),
    verbose=2
)
```

Evaluate Model

Plotting training and validation accuracy:

In [383]:

```
plot_accuracy(history)
```



Classification Report on validation data:

(validation data was used by the model for estimating performance, but model was not trained on it!)

In [384]:

```
preds = model.predict(X_valid, verbose=0)
# Classification report
print(classification_report(y_valid.argmax(axis=1), preds.argmax(axis=1)))

_, accuracy = model.evaluate(X_valid, y_valid, verbose=0)
print('Accuracy on validation: %.2f' % (accuracy*100))
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       335
           1       0.91      0.93      0.92       620
           2       0.99      0.95      0.97       335
           3       0.83      0.84      0.83       319
           4       0.96      0.92      0.94       454
           5       0.95      0.96      0.96       636

    accuracy                           0.93      2699
   macro avg       0.93      0.93      0.93      2699
weighted avg       0.93      0.93      0.93      2699

Accuracy on validation: 93.26
```

Classification Report on test data:

In [385]:

```
preds = model.predict(X_test, verbose=0)
# Classification report
print(classification_report(y_test.argmax(axis=1), preds.argmax(axis=1)))

_, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test: %.2f' % (accuracy*100))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       565
           1       0.92      0.94      0.93      1120
           2       0.96      0.97      0.97       536
           3       0.84      0.85      0.84       591
           4       0.94      0.95      0.95       818
           5       0.97      0.95      0.96      1133

    accuracy                           0.94      4763
   macro avg       0.94      0.94      0.94      4763
weighted avg       0.94      0.94      0.94      4763

Accuracy on test: 93.81
```

Seems like it help! f1-score of category #3 goes from 82-83 to 85. So we'll add it to our model :)

Notice: When you go thourgh this notebook, you'll notice that this feature is already included in the initial functions, but that was done **after** this section was completed and we saw that this feature actually help us

Cross Validation for hyper params

In [268]:

```
# Prepare data
X_train, X_test, y_train, y_test = get_data_for_model(test_size=0.2)
```

```
{'cakes_cupcakes_snack_cakes': 0, 'candy': 1, 'chips_pretzels_snacks': 2, 'chocolate': 3, 'cookies_b
iscuits': 4, 'popcorn_peanuts_seeds_related_snacks': 5}
Train:
        X_train: (25400, 16436) of float32
        y_train: (25400, 6) of float32
Test:
        X_test: (6351, 16436) of float32
        y_test: (6351, 6) of float32
```

In [269]:

```
# Early stopping callback for CV
es_cv = keras.callbacks.EarlyStopping(
    monitor='loss',
    mode='min',
    patience=10,  # stop training when there's no improvement in loss for 10 consecutive epochs
    restore_best_weights=True
)
```

In [270]:

```
def create_model_cv(dp=0.4, optimizer='adam', init='glorot_uniform', n_neurons=128):
    model_in = ks.Input(shape=(INPUT_SHAPE,), dtype='float32', sparse=True)
    out = ks.layers.Dense(n_neurons, activation='relu', kernel_initializer=init)(model_in)
    out = ks.layers.Dropout(dp)(out)
    out = ks.layers.Dense(n_neurons//2, activation='relu', kernel_initializer=init)(out)
    out = ks.layers.Dropout(dp)(out)
    out = ks.layers.Dense(n_neurons//8, activation='relu', kernel_initializer=init)(out)
    out = ks.layers.Dense(6, activation='softmax')(out)
    model = ks.Model(model_in, out)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy']) # rmsprop/adam
    return model
```

In [286]:

```
# Create KerasClassifier
INPUT_SHAPE = X_train.shape[1]
model_CV = KerasClassifier(
    model=create_model_cv,
    callbacks=[es_cv],
    loss="categorical_crossentropy",
    dp=None,
    optimizer=None,
    init=None,
    n_neurons=None
)
```

Define parameters for grid search

In [287]:

```
dp = [0.2, 0.4, 0.6]
optimizer = ["adam"]
batches = [16]
n_neurons = [128, 192, 256]
epochs = [3, 6, 10]

param_grid = dict(
    batch_size=batches,
    dp=dp,
    optimizer=optimizer,
    epochs=epochs,
    n_neurons=n_neurons
)

param_grid
```

Out[287]:

```
{'batch_size': [16],
 'dp': [0.2, 0.4, 0.6],
 'optimizer': ['adam'],
 'epochs': [3, 6, 10],
 'n_neurons': [128, 192, 256]}
```

In [288]:

```
grid = GridSearchCV(estimator=model_CV, param_grid=param_grid, cv=3, verbose=2)
```

In [ ]:

```
grid_result = grid.fit(X_train, y_train)
```

In [291]:

```
means = np.round(100*grid_result.cv_results_['mean_test_score'], 4)
stds = 100*grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

d = pd.DataFrame(params)
d['Mean'] = means
d['Std. Dev'] = stds

d.sort_values("Mean", ascending=False).drop(columns=["batch_size", "optimizer"])
```

Out[291]:

	dp	epochs	n_neurons	Mean	Std. Dev
25	0.6	10	192	93.3976	0.207484
24	0.6	10	128	93.3662	0.230753
19	0.6	3	192	93.3622	0.075154
9	0.4	3	128	93.3465	0.154075
20	0.6	3	256	93.3386	0.105761
23	0.6	6	256	93.3268	0.352725
22	0.6	6	192	93.3268	0.057542
21	0.6	6	128	93.2874	0.206028
26	0.6	10	256	93.2480	0.184312
18	0.6	3	128	93.2165	0.155946
0	0.2	3	128	93.1496	0.086465
15	0.4	10	128	93.0945	0.198523
13	0.4	6	192	93.0276	0.022079
10	0.4	3	192	93.0000	0.237690
2	0.2	3	256	92.9764	0.109951
17	0.4	10	256	92.9646	0.149537
16	0.4	10	192	92.9528	0.123617
12	0.4	6	128	92.9370	0.109145
14	0.4	6	256	92.8622	0.068063
11	0.4	3	256	92.8504	0.171585
1	0.2	3	192	92.8189	0.271695
3	0.2	6	128	92.7402	0.269879
4	0.2	6	192	92.7165	0.258555
8	0.2	10	256	92.6181	0.197847
6	0.2	10	128	92.6024	0.271513
7	0.2	10	192	92.4528	0.170228
5	0.2	6	256	92.4449	0.200110

Classification Report on test data:

In [292]:

```
preds = grid.predict(X_test)
# Classification report
report = classification_report(y_test.argmax(axis=1), preds.argmax(axis=1))
print(report)

print('Accuracy on test: %.2f' % round(100*accuracy_score(y_test, preds), 4))
```

```
397/397 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       744
           1       0.92      0.93      0.92      1508
           2       0.97      0.97      0.97       719
           3       0.82      0.86      0.84       765
           4       0.96      0.94      0.95      1093
           5       0.97      0.96      0.97      1522

    accuracy                           0.94      6351
   macro avg       0.94      0.94      0.94      6351
weighted avg       0.94      0.94      0.94      6351

Accuracy on test: 93.92
```

Before we continue, let's train this model on all data and save a prediction in the test set, in case this will be one of our final models:

In [293]:

```
train_data, y = load_train_data()
train_data, _, test_data = prepare_inputs(train_data, train_data, has_valid=False)

y, _ = prepare_targets(y, y, has_valid=False)

print("Train:")
print(f'\ttrain_data: {train_data.shape} of {train_data.dtype}')
print(f'\ty: {y.shape} of {y.dtype}')
print("Test:")
print(f'\ttest_data: {test_data.shape} of {test_data.dtype}')
```

```
{'cakes_cupcakes_snack_cakes': 0, 'candy': 1, 'chips_pretzels_snacks': 2, 'chocolate': 3, 'cookies_b
iscuits': 4, 'popcorn_peanuts_seeds_related_snacks': 5}
Train:
        train_data: (31751, 16444) of float32
        y: (31751, 6) of float32
Test:
        test_data: (3525, 16444) of float32
```

In [295]:

```
grid.best_params_
```

Out[295]:

```
{'batch_size': 16,
 'dp': 0.6,
 'epochs': 10,
 'n_neurons': 192,
 'optimizer': 'adam'}
```

In [ ]:

```
# Fit model on all train data
INPUT_SHAPE = train_data.shape[1]
model_CV_2 = create_model_cv(
    dp=0.6,
    optimizer='adam',
    n_neurons=192
)
history = model_CV_2.fit(
    train_data,
    y,
    epochs=10,
    batch_size=16,
    verbose=2
)
```

In [251]:

```
grid.best_params_
```

Out[251]:

```
{'batch_size': 16,
 'dp': 0.6,
 'init': 'glorot_uniform',
 'n_neurons': 256,
 'optimizer': 'rmsprop'}
```

Sanity check on train data

In [300]:

```
preds = model_CV_2.predict(train_data)
# Classification report
print(classification_report(y.argmax(axis=1), preds.argmax(axis=1)))
accuracy = sum(preds.argmax(axis=1) == y.argmax(axis=1)) / len(preds)
print('Accuracy on train: %.2f' % (accuracy*100))
```

```
993/993 [==============================] - 2s 2ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3786
           1       0.99      0.95      0.97      7584
           2       0.99      1.00      1.00      3680
           3       0.90      0.98      0.94      3772
           4       0.99      0.99      0.99      5284
           5       1.00      0.99      0.99      7645

    accuracy                           0.98     31751
   macro avg       0.98      0.99      0.98     31751
weighted avg       0.98      0.98      0.98     31751

Accuracy on train: 98.23
```

Predict on test data and save results

In [301]:

```
preds = model_CV_2.predict(test_data).argmax(axis=1)
preds = [mapping[i] for i in preds]
test_indexes = load_test_data().idx
```

```
111/111 [==============================] - 0s 2ms/step
```

In [302]:

```
results_model_02 = pd.DataFrame({"idx": indexes, "pred_cat": preds})
results_model_02.to_csv("model_02.csv", index=False)

# For sanity check
t = load_test_data()
t["pred_cat"] = preds
t.to_csv("model_02_check.csv", index=False)
```

Conclusions

Well, seems like the different hyperparameters don't affect that much the network's performance, as we see the accuracy for all of them is not very different, and there's no clear answer whether it improved the model.

Adding the predictions from our CNN

We decided to seperatly train our CNN on the images, and add its predictions as another feature for our tabular data. We understand that its both not the most correct and elegant way to combine them, and the better way would be to combine 2 networks together and add some dense layers after that, and also that it might cause us to overfit on the training data. We chose this way beacuse of time constrains - we tried buiding a combined NN but we got a little bit confused and things didn't work for us. We'll be hyper aware of overfitting when we check if this new feature is helpful :)

First step we did was add the "with_cnn" flag to all relevant functions, so that if we choose to add this feature to the model, we'll be able to.

But, in order to fairly check whether this feature improves the model, we did the following:

- Trained the CNN on 70% of the data, left 30% for test.
- Saved the indexes we used for test, and seperated our tabular data exactly the same - so we won't predict on products which their image we used as a training image.

Therefore, the following code is to manually create these dfs. It's a bit messy and we apologize for it :). If we had more time we would better integrate this as part of the already created functions.

(Code is hidden, appears in the notebook)

```
{'cakes_cupcakes_snack_cakes': 0, 'candy': 1, 'chips_pretzels_snacks': 2, 'chocolate': 3, 'cookies_b
iscuits': 4, 'popcorn_peanuts_seeds_related_snacks': 5}
```

In [200]:

```
# Sanity check
print("Train:")
print(f'\tX_train_cnn: {X_train_cnn.shape} of {X_train_cnn.dtype}')
print(f'\ty_train_cnn: {y_train_cnn.shape} of {y_train_cnn.dtype}')
print("Validation:")
print(f'\tX_valid_cnn: {X_valid_cnn.shape} of {X_valid_cnn.dtype}')
print(f'\ty_valid_cnn: {y_valid_cnn.shape} of {y_valid_cnn.dtype}')
print("Test:")
print(f'\tX_test_cnn: {X_test_cnn.shape} of {X_test_cnn.dtype}')
print(f'\ty_test_cnn: {y_test_cnn.shape} of {y_test_cnn.dtype}')
```

```
Train:
        X_train_cnn: (22228, 16442) of float32
        y_train_cnn: (22228, 6) of float32
Validation:
        X_valid_cnn: (3174, 16442) of float32
        y_valid_cnn: (3174, 6) of float32
Test:
        X_test_cnn: (6349, 16442) of float32
        y_test_cnn: (6349, 6) of float32
```

In [201]:

```
# Early stopping callback
es_cnn = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    mode='min',
    patience=15,  # stop training when there's no improvement in val_loss for 15 consecutive epochs
    restore_best_weights=True
)
```

In [202]:

```
# Same model we used in our first model
def create_model_cnn(dp=0.4, optimizer='adam', num_nuerons=[128, 32, 16]):
    model_in = ks.Input(shape=(INPUT_SHAPE,), dtype='float32', sparse=True)
    out = ks.layers.Dense(num_nuerons[0], activation='relu')(model_in)
    out = ks.layers.Dropout(dp)(out)
    out = ks.layers.Dense(num_nuerons[1], activation='relu')(out)
    out = ks.layers.Dropout(dp)(out)
    out = ks.layers.Dense(num_nuerons[2], activation='relu')(out)
    out = ks.layers.Dense(6, activation='softmax')(out)
    model = ks.Model(model_in, out)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy']) # rmsprop/adam
    return model
```

In [ ]:

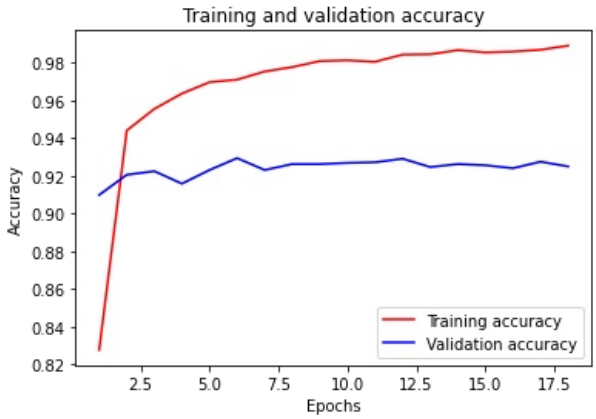
```
# Create model
INPUT_SHAPE = X_train_cnn.shape[1]
model = create_model_cnn()

# Fit model on training data
history = model.fit(
    X_train_cnn,
    y_train_cnn,
    callbacks=[es_cnn],
    epochs=10000,
    shuffle=True,
    validation_data=(X_valid_cnn, y_valid_cnn),
    verbose=2
)
```

Evaluate Model

Plotting training and validation accuracy:

In [204]:

```
plot_accuracy(history)
```



Classification Report on validation data:

(validation data was used by the model for estimating performance, but model was not trained on it!)

In [205]:

```
preds = model.predict(X_test_cnn, verbose=0)
# Classification report
print(classification_report(y_test_cnn.argmax(axis=1), preds.argmax(axis=1)))
_, accuracy = model.evaluate(X_test_cnn, y_test_cnn, verbose=0)
print('Accuracy on test: %.2f' % (accuracy*100))
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96      1079
           1       0.87      0.90      0.89      1509
           2       0.94      0.95      0.94       648
           3       0.74      0.73      0.73       758
           4       0.92      0.90      0.91       981
           5       0.93      0.93      0.93      1374

    accuracy                           0.90      6349
   macro avg       0.89      0.89      0.89      6349
weighted avg       0.90      0.90      0.90      6349

Accuracy on test: 89.84
```

Looks like we do much worse here, so we're not gonna include it in one of our predictions :(

We think it's because our CNN model does very poorly on chocolate, but pretty good on all other categories. So what happens is this model believes our CNN prediction as generally it's a good one, and therefore performs now poorly on chocolate as well.

We have ideas our to improve this:

- Perform CV with this feature as well.
- Engineer this feature in another way - maybe ignore the CNNs prediction for chocolate, and create less features that indicate whether our CNN thinks the product belongs to some other category (but only categories it performs well on).
- Improve our CNN, of course.
- Engineer additional features for chocolate that maybe will help balance the CNNs predictions.

We just don't have time to do these, but we wanted to mention them.

another meme