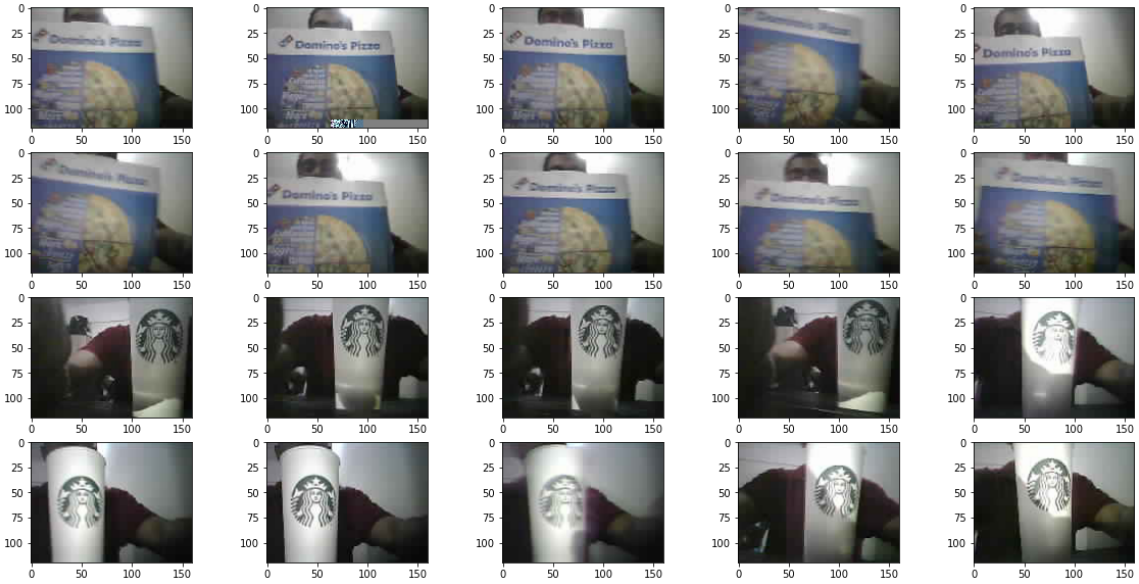
In [3]:

```
"""
Collect images from Esp32-cam web server
"""
from logging import basicConfig, INFO
from everywhereml.data import ImageDataset
from everywhereml.data.collect import MjpegCollector
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
# you need to manually create this folder in the current working directory
base_folder = 'fnb_1'
# copy here the address printed on the Serial Monitor
# (the one after "MJPEG stream available at")
IP_ADDRESS_OF_ESP = 'http://esp32cam.local:81'
basicConfig(level=INFO)

try:
    # if our dataset folder already exists, load it
    image_dataset = ImageDataset.from_nested_folders(
        name='fnb_1',
        base_folder=base_folder
    )
except FileNotFoundError:
    # if the dataset folder does not exists, collect the samples
    # from the Esp32-cam web server
    # duration is how long (in seconds) the program will collect
    # the images for each class
    #
    # After each class collection, you may need to manually create the
    # subfolder to store the class images.
    #
    # Follow the instructions accurately!
    mjpeg_collector = MjpegCollector(address=IP_ADDRESS_OF_ESP)
    image_dataset = mjpeg_collector.collect_many_classes(
        dataset_name='fnb_1',
        base_folder=base_folder,
        duration=30
    )

print(image_dataset)
```

```
ImageDataset[fnb_1](num_images=941, num_labels=2, labels=['Dominoes', 'Sta
rbucks'])
```

In [4]:

```
image_dataset.preview(
    samples_per_class=10,
    rows_per_class=2,
    figsize=(20, 10)
)
```

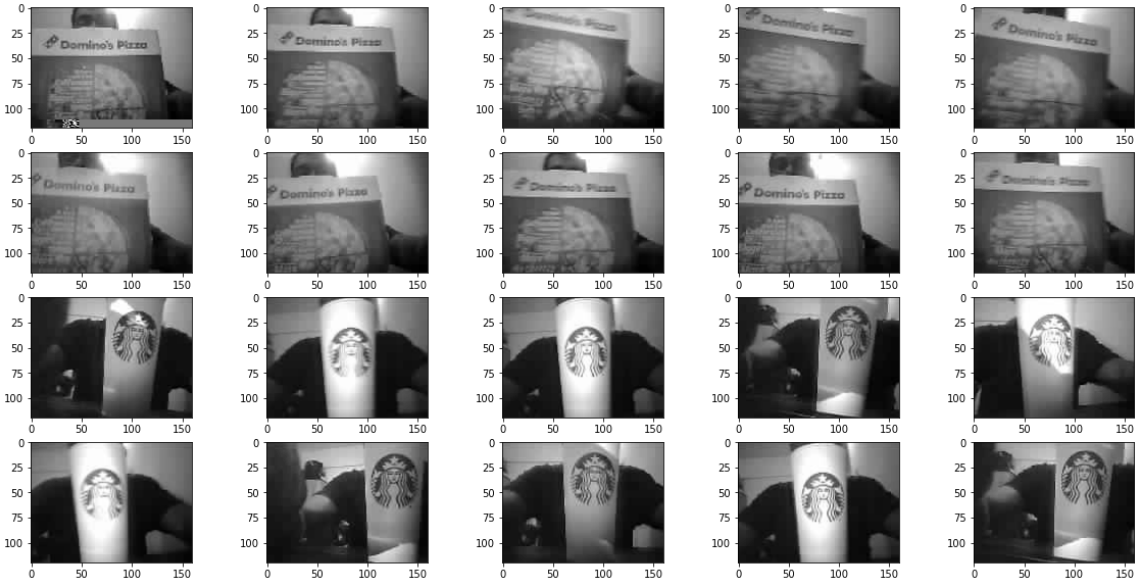


In [5]:

```
"""
Image classification with HOG works on grayscale images at the moment
So convert images to grayscale in the range 0-255
"""
image_dataset = image_dataset.gray().uint8()
```

In [6]:

```
"""
Preview grayscale images
"""
image_dataset.preview(
    samples_per_class=10,
    rows_per_class=2,
    figsize=(20, 10),
    cmap='gray'
)
```



In [7]:

```
"""
Create an image recognition pipeline with HOG feature extractor
"""
from everywhereml.preprocessing.image.object_detection import HogPipeline
from everywhereml.preprocessing.image.transform import Resize

pipeline = HogPipeline(
    transforms=[
        Resize(width=40, height=30)
    ]
)

# Convert images to feature vectors
feature_dataset = pipeline.fit_transform(image_dataset)
feature_dataset.describe()
```

```
HOG: 100%|██████████| 941/941 [00:01<00:00, 742.74it/s]
```

Out[7]:

	hog0	hog1	hog2	hog3	hog4	hog5	hog6
count	941.000000	941.000000	941.000000	941.000000	941.000000	941.000000	941.000000
mean	0.252473	0.170117	0.128185	0.227171	0.129601	0.047671	0.044830
std	0.315454	0.168568	0.129483	0.258759	0.196648	0.073353	0.081300
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.035604	0.048629	0.045200	0.024356	0.000000	0.000000	0.000000
50%	0.099504	0.119377	0.089593	0.155397	0.042005	0.007275	0.000000
75%	0.387834	0.216771	0.172035	0.313401	0.168076	0.072461	0.066358
max	1.000000	1.000000	1.000000	1.000000	1.000000	0.491762	0.664836

8 rows × 136 columns

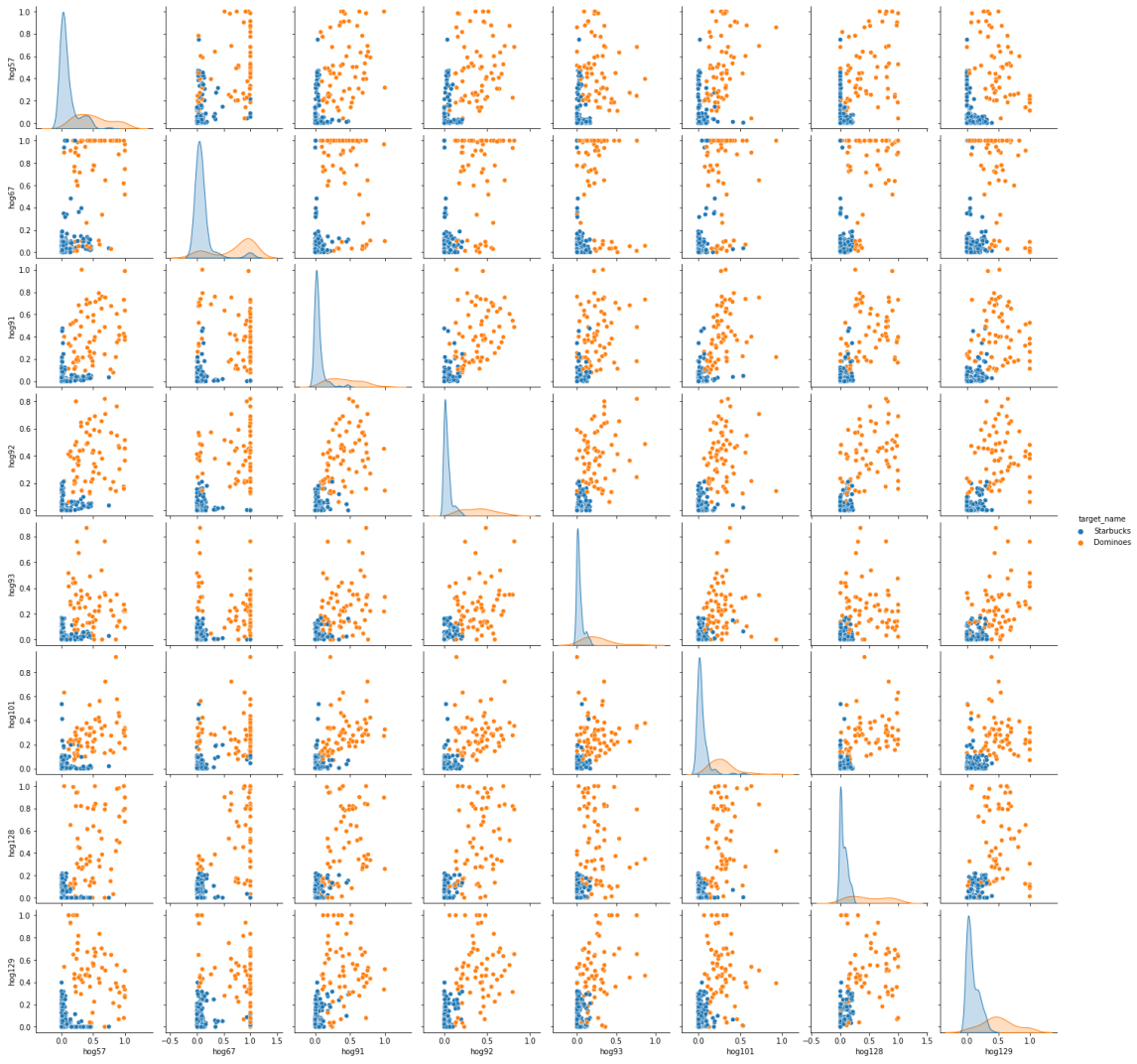
In [8]:

```
"""
Print pipeline description
"""
print(pipeline)
```

```
ImagePipeline: HogPipeline
---------
 - Resize(from=(160, 120), to=(40, 30), pixformat=gray)
 > HOG(block_size=8, bins=9, cell_size=3)
```

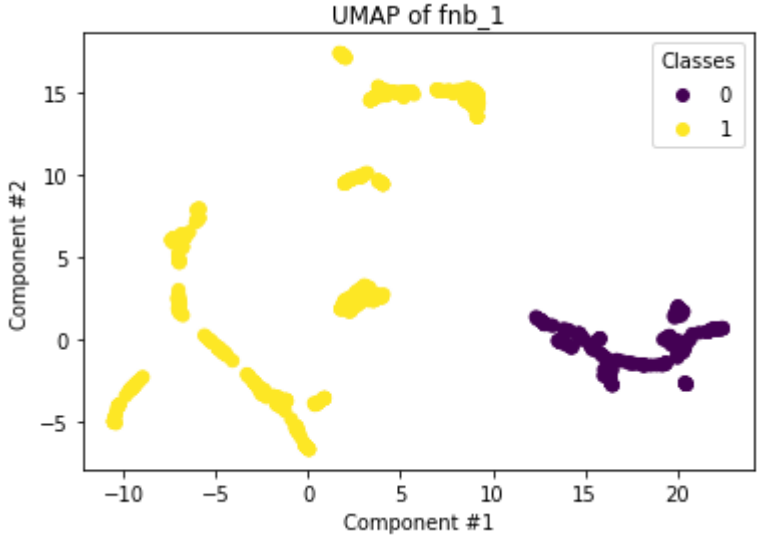
In [9]:

```
"""
Plot pairplot of features.
Feel free to open the image in a new window to see it at full scale.
In the next line:
 - n is the number of points to plot (the greater the value, the longer it takes)
 - k is the number of features (values greater than 10 become messy)
"""
feature_dataset.plot.features_pairplot(n=200, k=8)
```



In [10]:

```
"""
Plot UMAP of features
If features are discriminative, we should see well defined clusters of points
"""
feature_dataset.plot.umap()
```



In [11]:

```
"""
Create and fit RandomForest classifier
"""
from everywhereml.sklearn.ensemble import RandomForestClassifier

for i in range(10):
    clf = RandomForestClassifier(n_estimators=5, max_depth=10)

    # fit on train split and get accuracy on the test split
    train, test = feature_dataset.split(test_size=0.4, random_state=i)
    clf.fit(train)

    print('Score on test set: %.2f' % clf.score(test))

# now fit on the whole dataset
clf.fit(feature_dataset)
```

```
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
Score on test set: 1.00
```

Out[11]:

```
RandomForestClassifier(base_estimator=DecisionTreeClassifier(), bootstrap=
True, ccp_alpha=0.0, class_name=RandomForestClassifier, class_weight=None,
criterion=gini, estimator_params=('criterion', 'max_depth', 'min_samples_s
plit', 'min_samples_leaf', 'min_weight_fraction_leaf', 'max_features', 'ma
x_leaf_nodes', 'min_impurity_decrease', 'random_state', 'ccp_alpha'), max_
depth=10, max_features=auto, max_leaf_nodes=None, max_samples=None, min_im
purity_decrease=0.0, min_samples_leaf=1, min_samples_split=2, min_weight_f
raction_leaf=0.0, n_estimators=5, n_jobs=None, num_outputs=2, oob_score=Fa
lse, package_name=everywhereml.sklearn.ensemble, random_state=None, templa
te_folder=everywhereml/sklearn/ensemble, verbose=0, warm_start=False)
```

In [12]:

```
"""
Export pipeline to C++
Replace the path to your actual sketch path
"""
print(pipeline.to_arduino_file(
    filename=r'C:\Users\hp\Documents\Arduino\LogoDetection\HogPipeline.h',
    instance_name='hog'
))
```

```
#ifndef UUID2379310971056
#define UUID2379310971056


    #ifndef UUID2379310971872
#define UUID2379310971872

/**
  * HOG(block_size=8, bins=9, cell_size=3)
 */
class HOG {
    public:

        /**
         * Transform input image
         */
         template<typename T, typename U>
        bool transform(T *input, U *output) {
```

In [13]:

```
"""
Export classifier to C++
Replace the path to your actual sketch path

The class_map parameters convert numeric classes to human-readable strings
"""
print(clf.to_arduino_file(
    filename=r'C:\Users\hp\Documents\Arduino\LogoDetection\HogClassifier.h',
    instance_name='classifier',
    class_map=feature_dataset.class_map
))
```

```
#ifndef UUID2377304400800
#define UUID2377304400800

/**
  * RandomForestClassifier(base_estimator=DecisionTreeClassifier(), boo
tstrap=True, ccp_alpha=0.0, class_name=RandomForestClassifier, class_we
ight=None, criterion=gini, estimator_params=('criterion', 'max_depth',
'min_samples_split', 'min_samples_leaf', 'min_weight_fraction_leaf', 'm
ax_features', 'max_leaf_nodes', 'min_impurity_decrease', 'random_stat
e', 'ccp_alpha'), max_depth=10, max_features=auto, max_leaf_nodes=None,
max_samples=None, min_impurity_decrease=0.0, min_samples_leaf=1, min_sa
mples_split=2, min_weight_fraction_leaf=0.0, n_estimators=5, n_jobs=Non
e, num_outputs=2, oob_score=False, package_name=everywhereml.sklearn.en
semble, random_state=None, template_folder=everywhereml/sklearn/ensembl
e, verbose=0, warm_start=False)
 */
class RandomForestClassifier {
    public:

        /**
```

In [ ]: