classifying MNIST dataset

dataset

MNIST with 0-padding (28x28 to 32x32)

```
(mnist_train_images, mnist_train_labels), (mnist_test_images, mnist_test_labels) = keras.datasets.mnist.load_data()

def resize_batch(imgs):
    # A function to resize a batch of MNIST images to (32, 32)
    imgs = imgs.reshape((-1, 28, 28, 1))
    resized_imgs = np.zeros((imgs.shape[0], 32, 32, 1))
    for i in range(imgs.shape[0]):
        resized_imgs[i, ..., 0] = transform.resize(imgs[i, ..., 0], (32, 32))
    return resized_imgs

# Resize MNIST images
mnist_train_images = resize_batch(mnist_train_images)
mnist_test_images = resize_batch(mnist_test_images)
```

shape of training data: (60000, 32, 32, 1)

shape of testing data: (10000, 32, 32, 1)

cwSaab

```
def Shrink(X, shrinkArg):
    win = shrinkArg['win']
    X = view_as_windows(X, (1,win,win,1), (1,win,win,1))
    return X.reshape(X.shape[0], X.shape[1], X.shape[2], -1)
```

```
X = mnist_train_images

SaabArgs = [{'num_AC_kernels': -1, 'needBias': True, 'useDC': False, 'batch': None}]

print("Testing cwSaab with depth=1")
cwsaab = cwSaab(depth=1, energyTH=0.5, SaabArgs=SaabArgs, shrinkArgs=shrinkArgs,
                concatArg=concatArg, splitMode=0, cwHop1=True)

output = cwsaab.fit(X)
output = cwsaab.transform(X)
```

DFT

adjust 'thrs' to determine what percentage of the features we want to keep

```
features = output[0].reshape(len(X), -1)
labels = mnist_train_labels
selected, dft_loss = feature_selection(features, labels, FStype='DFT_entropy', thrs=0.8, B=16)
print("Selected features:", selected)
```

XGBoost

parameters: n_estimators, eta, max_depth

```
model = XGBClassifier(
    booster='gbtree',
    objective='multi:softprob', # multi-class classification
    n_estimators=100,           # number of estimators
    num_class=10,               # number of classes
    eta=0.3,                    # learning rate
    max_depth=6,                # maximum depth of the trees
    eval_metric='mlogloss',     # evaluation metric
    use_label_encoder=False     # to suppress a warning
)
model.fit(X_train, y_train)
```

Score

Make predictions using test data and calculate accuracy

```
# Make predictions
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"MNIST Accuracy: {accuracy * 100:.2f}%")
```

```
MNIST Accuracy: 98.06%
```

Problems

cwSaab

1. max-pooling implementation
2. concatenating implementation