```
import os
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from tqdm import tqdm
import joblib
from sklearn.model_selection import GridSearchCV
import cv2
import seaborn as sns
import time
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

train_data = os.getcwd() + "/drive/MyDrive/train"

train_images = []
for root, _, files in os.walk(train_data): # Use os.walk to traverse all subdirectories
    for file in files:
        if file.endswith(('.jpg', '.jpeg', '.png')): # Filter for common image file extensions
            train_images.append(os.path.join(root, file))

features = []
labels = []
image_size = (50, 50)
num_processed = 0  # Keep track of processed images

# Process train images
for image_path in tqdm(train_images, desc="Processing Train Images"):
    if 'cat' in os.path.basename(image_path): # Check for 'cat' in the filename
        label = 0
    else:
        label = 1

    image_read = cv2.imread(image_path)
    if image_read is None:
        print(f"Failed to read image: {image_path}")
        continue

    image_resized = cv2.resize(image_read, image_size)
    image_normalized = image_resized / 255.0
    image_flatten = image_normalized.flatten()
    features.append(image_flatten)
    labels.append(label)
    num_processed += 1

print("Number of images processed:", num_processed)
print(features)
print(labels)
```

```
    , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
del train_images
```

```
features = np.asarray(features)
labels = np.asarray(labels)

# train test split
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, shuffle=True, random_state=42)
```

```
del features
del labels
```

Double-click (or enter) to edit

```
from sklearn.preprocessing import StandardScaler
```

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC())
])

# Define hyperparameter grid
param_grid = {
    'svm__kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'svm__C': [0.1, 1, 10],
}

#starting time for training
start_time = time.time()

grid_search = GridSearchCV(pipeline, param_grid, cv=3, verbose=4)
grid_search.fit(X_train, y_train)

# ending time for training
end_time = time.time()
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
[CV 1/3] END ....svm__C=0.1, svm__kernel=linear;, score=0.576 total time= 1.5min
[CV 2/3] END ....svm__C=0.1, svm__kernel=linear;, score=0.579 total time= 1.6min
[CV 3/3] END ....svm__C=0.1, svm__kernel=linear;, score=0.578 total time= 1.5min
[CV 1/3] END .......svm__C=0.1, svm__kernel=rbf;, score=0.569 total time= 1.7min
[CV 2/3] END .......svm__C=0.1, svm__kernel=rbf;, score=0.565 total time= 1.7min
[CV 3/3] END .......svm__C=0.1, svm__kernel=rbf;, score=0.574 total time= 1.8min
[CV 1/3] END ......svm__C=0.1, svm__kernel=poly;, score=0.555 total time= 1.7min
[CV 2/3] END ......svm__C=0.1, svm__kernel=poly;, score=0.555 total time= 1.7min
[CV 3/3] END ......svm__C=0.1, svm__kernel=poly;, score=0.555 total time= 1.7min
[CV 1/3] END ...svm__C=0.1, svm__kernel=sigmoid;, score=0.549 total time= 1.6min
[CV 2/3] END ...svm__C=0.1, svm__kernel=sigmoid;, score=0.549 total time= 1.6min
[CV 3/3] END ...svm__C=0.1, svm__kernel=sigmoid;, score=0.564 total time= 1.6min
[CV 1/3] END ......svm__C=1, svm__kernel=linear;, score=0.576 total time= 1.5min
[CV 2/3] END ......svm__C=1, svm__kernel=linear;, score=0.579 total time= 1.7min
[CV 3/3] END ......svm__C=1, svm__kernel=linear;, score=0.578 total time= 1.5min
[CV 1/3] END .........svm__C=1, svm__kernel=rbf;, score=0.657 total time= 1.7min
[CV 2/3] END .........svm__C=1, svm__kernel=rbf;, score=0.680 total time= 1.7min
[CV 3/3] END .........svm__C=1, svm__kernel=rbf;, score=0.663 total time= 1.7min
[CV 1/3] END ........svm__C=1, svm__kernel=poly;, score=0.557 total time= 1.7min
```

```
[CV 2/3] END ........svm__C=1, svm__kernel=poly;, score=0.582 total time= 1.7min
[CV 3/3] END ........svm__C=1, svm__kernel=poly;, score=0.580 total time= 1.7min
[CV 1/3] END .....svm__C=1, svm__kernel=sigmoid;, score=0.520 total time= 1.2min
[CV 2/3] END .....svm__C=1, svm__kernel=sigmoid;, score=0.549 total time= 1.2min
[CV 3/3] END .....svm__C=1, svm__kernel=sigmoid;, score=0.559 total time= 1.1min
[CV 1/3] END .....svm__C=10, svm__kernel=linear;, score=0.576 total time= 1.6min
[CV 2/3] END .....svm__C=10, svm__kernel=linear;, score=0.579 total time= 1.7min
[CV 3/3] END .....svm__C=10, svm__kernel=linear;, score=0.578 total time= 1.5min
[CV 1/3] END ........svm__C=10, svm__kernel=rbf;, score=0.655 total time= 1.8min
[CV 2/3] END ........svm__C=10, svm__kernel=rbf;, score=0.665 total time= 1.8min
[CV 3/3] END ........svm__C=10, svm__kernel=rbf;, score=0.664 total time= 1.8min
[CV 1/3] END .......svm__C=10, svm__kernel=poly;, score=0.583 total time= 1.7min
[CV 2/3] END .......svm__C=10, svm__kernel=poly;, score=0.609 total time= 1.6min
[CV 3/3] END .......svm__C=10, svm__kernel=poly;, score=0.590 total time= 1.6min
[CV 1/3] END ....svm__C=10, svm__kernel=sigmoid;, score=0.502 total time=  54.0s
[CV 2/3] END ....svm__C=10, svm__kernel=sigmoid;, score=0.534 total time=  51.2s
[CV 3/3] END ....svm__C=10, svm__kernel=sigmoid;, score=0.558 total time=  49.7s
```

```
del X_train
del y_train
```

```
best_pipeline = grid_search.best_estimator_
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters: ", best_params)
print("Best Score: ", best_score)
```

```
Best Parameters:  {'svm__C': 1, 'svm__kernel': 'rbf'}
Best Score:  0.6667531321302991
```

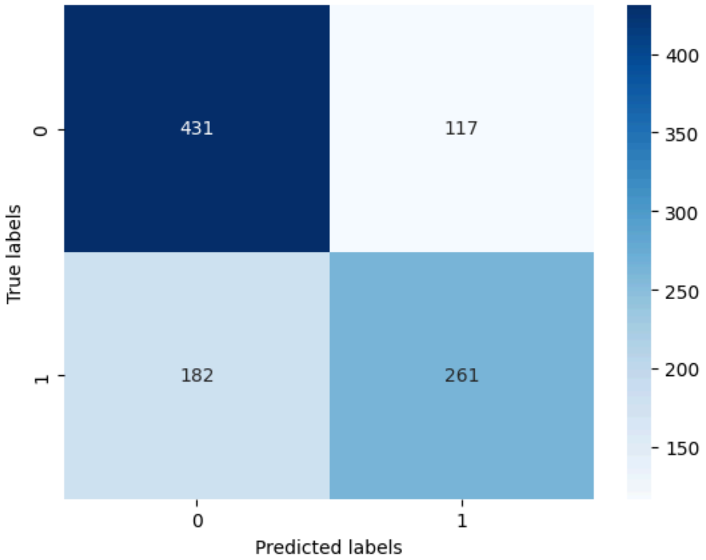
```
# Evaluation on test dataset
accuracy = best_pipeline.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.698284561049445
```

```
y_pred = best_pipeline.predict(X_test)

# classification report
target_names = ['Cat', 'Dog']
svm_rep = classification_report(y_test, y_pred, target_names=target_names)
print("Classification Report:\n", svm_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

         Cat       0.70      0.79      0.74       548
         Dog       0.69      0.59      0.64       443

    accuracy                           0.70       991
   macro avg       0.70      0.69      0.69       991
weighted avg       0.70      0.70      0.69       991
```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```



Start coding or generate with AI.