Image Processing – Prediction

The goal of this challenge is to use machine learning to predict the calcification of cat and dog images.

Image processing is a technique for performing various operations on a picture. To obtain a greater image quality. I'll use machine learning to determine whether an image has a cat or a dog.

I did some data exploration after reading the file. I also discovered that the dataset is perfectly balanced.

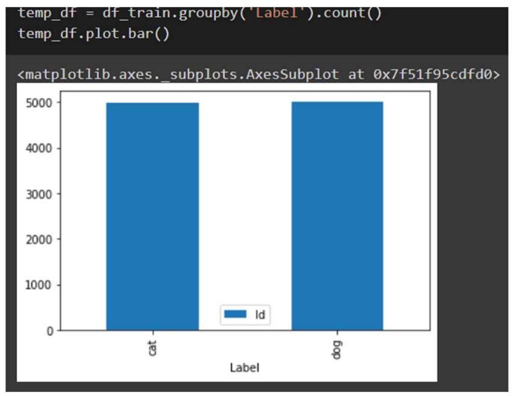

Figure 1 Summarizing the label dataset

After loading the image, resize it with three parameters: ids- list of images, folder location, and the primary parameter dim, which sets the image's dimensions.

```
def load_images(ids, folder_path, dim):
  images = []
  for id in ids:
    image_path = os.path.join(folder_path, "{}.jpg".format(id))
    img = cv2.imread(image_path)

    # Resize if necessary
    if img.shape[0] != dim[1] or img.shape[1] != dim[0]:
      img = cv2.resize(img, dim)
    images.append(img)
  return images
```

Figure 2 Resizing the image

Once the images have been resized, base dim is set, and the data is split into training and testing (in this case, 10000 images for training and 1000 images for testing).

```
base_dim = (200, 200)

# load train images
train_image_folder = "/content/data/train_images"
train_images = load_images(df_train['Id'], train_image_folder, base_dim)
print(f'Number of training images loaded: {len(train_images)}')

# load test images
test_image_folder = "/content/data/test_images"
test_images = load_images(df_test['Id'], test_image_folder, base_dim)
print(f'Number of testing images loaded: {len(test_images)}')
```

```
Number of training images loaded: 10000
Number of testing images loaded: 1000
```

Figure 3 Splitting images for training and testing

Model Generation: M1 Created the svm classifier model with 100 random states and a linear kernel.

```
# method to train and predict using SVM classifier
def get_svm_predictions(X_train, X_val, y_train, y_val):
  # build model
  clf = svm.SVC(kernel='linear', random_state=100)
  clf.fit(X_train, y_train)

  # Make predictions on test data
  y_pred = clf.predict(X_val)

  # evaluation
  accuracy, confusion_matrix = evaluate(y_val, y_pred)
  print(f'Accuracy: {accuracy}')
  plot_confusion_matrix(confusion_matrix)

  return clf
```

Figure 4 Model Generation

Method to get image features

Preprocessing - gray scaling
Grayscale is a range of monochromatic shades from black to white.
Gray scaling in converting images to grayscale.
Features - image vector
The simplest method to convert an image to a vector is matrix vectorisation. For N x M matrix, a vector of length NM will be created
Algorithm – SVM

```python
# method to get image features
def get_features_m1(images):
  features_list = []
  for img in images:
    # image preprocessing
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # vectorise/ feature extraction
    features = img_grayscaled.flatten()

    features_list.append(features)

  features_list = np.array(features_list)
  return features_list
```

Figure 5 Setting image features

```python
# feature extraction
features_train = get_features_m1(train_images)
print(features_train.shape)

# data split for train and validation
X_train, X_val, y_train, y_val = train_test_split(features_train, df_train['Label'], test_size=0.3, random_state=100)

# train and validation
model = get_svm_predictions(X_train, X_val, y_train, y_val)
```

```
(10000, 40000)
Accuracy: 0.516
```

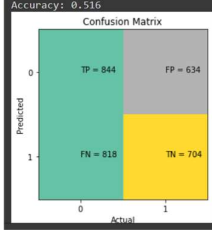

Figure 6 Train and validation model

```python
# get model predictions
predictions = model.predict(features_test)
print(predictions)
```

```
(1000, 40000)
[0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1
 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0
 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1
 1 1 1 1 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1
 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1
 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 1 1 1 0
 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 1
 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1
 1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 1 1 0 0 1 0 0 0
 0 0 0 0 0 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 0 0
 1 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 0 1
 1 0 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1 0 0 0
 0 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0
 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 1
 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 1 0
 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0
 1 1 1 0 1 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0
 1 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0
 1 1 0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 1
 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 1
 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0
 1 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0
```

Figure 7 Predicting on test images

M4

- Preprocessing - gray scaling, smoothing
- Features - HOG features
- Algorithm – SVM
- The **HOG feature descriptor** counts the occurrences of gradient orientation in localized portions of an image as features.

```python
# method to get image features
def get_features_m4(images):
  features_list = []
  for img in images:
    # image preprocessing
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Resize image if necessary
    img_resized = cv2.resize(img_grayscaled, (64, 128))

    # feature extraction
    features, hog_image = hog(img_resized, orientations=9, pixels_per_cell=(8, 8),
                    cells_per_block=(2, 2), visualize=True)

    features_list.append(features)

  features_list = np.array(features_list)
  return features_list
```

Figure 8 Model 4

```python
# feature extraction
features_train = get_features_m4(train_images)
print(features_train.shape)

# data split for train and validation
X_train, X_val, y_train, y_val = train_test_split(features_train, df_train['genus'], test_size=0.3, random_state=100)

# train and validation
model = get_svm_predictions(X_train, X_val, y_train, y_val)
```

Figure 9 Feature extraction and splitting

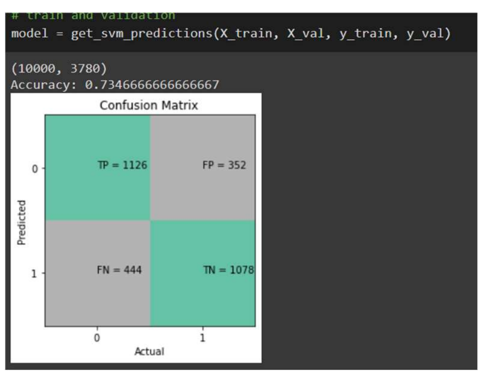

Figure 10 Model Accuracy

```
# get model predictions
predictions = model.predict(features_test)
print(predictions)
```

```
(1000, 3780)
[1 0 0 0 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1
 1 1 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 0
 1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 1
 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 1 0
 0 1 0 0 1 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0
 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 1 1 0 1 1 1 1 1 0 1
 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 1 1
 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 0
 0 0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1
 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 0
 1 1 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1
 1 0 0 0 0 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 0 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1
 0 0 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 0
 1 0 0 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1
 1 1 1 0 0 0 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 0 1 1 1 1 0 1
 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 1 1 0
 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1 1 1 1
 1 1 0 1 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1
 1 0 1 1 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1
 0 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0
 0 0 1 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 0 1 0 1 1 1 1
 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 1 1 1 0 1 1 0 1 1 1 1 0 1
 0 0 1 1 0 0 0 1 0 0 1 1 1 1 0 0 0 1 1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 0
```

Figure 11 Prediction values

```
# make predictions using the best model
predictions = model.predict(features_test)
```

```
# add predictions to the 'Label' column
df_test['Label'] = predictions
```

```
# save data frame to .csv file
df_test.to_csv('/content/test-predictions.csv', index=False)
```

*Figure 12	Predicting the test image*

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x=df_test['Label'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1e2debf450>
```

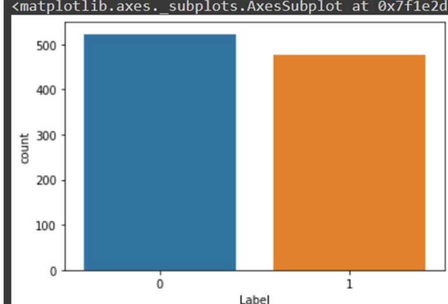

Figure 13 Test image predicting in graphs

Summarizing

Model	Preprocessing	Features	Accuracy
M1	Gray scaling	Image vector	0.657
M2	Gray scaling, smoothing	Image vector	0.639
M3	Gray scaling	Edge map to vector	0.608
M4	Gray scaling	Hog features	0.734

Conclusion: Using svm machine learning, I learned and implemented image processing. When several SVM models are applied in image processing and feature extraction, M4's model accuracy improves to 73.4% using RBF kernel.