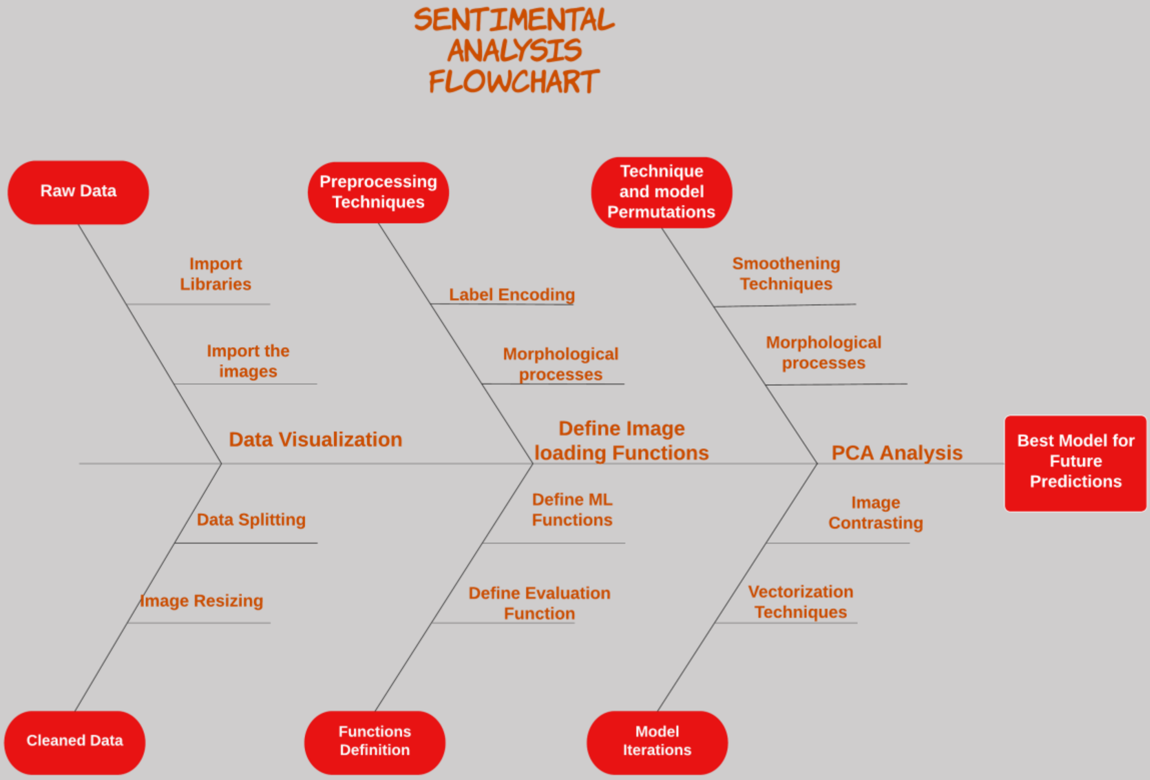
Feline vs. Canine:
The Ultimate Showdown of Image Classification

"Have you ever stared at a picture of a fluffy animal, wondering if it's a dog or a cat? Don't worry, it's a classic case of 'paw confusion'. Fear not, because with the power of machine learning, we can finally settle this age-old debate and bring peace to the internet. So, sit back, relax, and let's embark on a journey of fur-filled fun as we explore the world of dog and cat image classification!"



This blog explains how image classification algorithms work, eliminating debates with friends over whether that fluffy creature is a feline or a canine. Click Here for colab Notebook.

10000 train images and 1000 test images were used. A breakdown:

Machine Learning models.

- SVM (Linear, poly, RBF, Sigmoid)
- Random Forest Classifier

Preprocessing Techniques

- Gaussian Filtering
- Bilateral Filtering
- Adaptive Thresholding
- Morphological Operations to remove small objects.

Feature Extraction Techniques

- Image Vector
- Edge map to Vector
- HOG Vectorization

Principal Component Analysis

- Iterations from n_components = 1000
- Optimal number discovered was 107.

13 different models were tested. Screenshots are displayed in subsequent sections.

Observations

1. Gaussian smoothing had no effect on accuracy until contrasting techniques were applied.
2. Best performing SVM kernel is RBF because it captures complex and non-linear relationships between input features and output classes, making it more accurate in classifying images.
3. The HOG performed better than other vectorization techniques.
4. Bilateral filtering did not improve accuracy.
5. PCA improved the accuracy of the model by reducing the image features.
6. Adaptive thresholding and morphing techniques improved accuracy significantly.

Results: The highest accuracy achieved is 0.771(M13)

Summary

Model	Preprocessing	Vectorization Technique	ML Model	PCA	Accuracy
M1	gray scaling	Image Vector	SVM - Linear	No	0.516
M2	gray scaling	Image Vector	SVM - rbf	No	0.641
M3	gray scaling	Image Vector	SVM poly	No	0.599
M4	gray scaling	Image Vector	SVM - Sigmoid	No	0.527
M5	gray scaling, smoothing	Image Vector	SVM - rbf	No	0.64
M6	gray scaling	Edge map to Vector	SVM - rbf	No	0.608
M7	gray scaling	HOG	SVM - rbf	No	0.735
M8	gray scaling	Image Vector and Edge Map to Vector	SVM - linear	No	0.57866
M9	gray scaling, Bilateral filtering	HOG	SVM - rbf	No	0.735
M10	gray scaling	HOG	Random Forest	No	0.701
M11	gray scaling	HOG	SVM - rbf	Yes	0.736
M12	gray scaling	HOG	SVM - rbf	Yes(n_components = 107)	0.752
M13	gray scaling, Gaussian smoothing, adaptive thresholding, Morphology	HOG	SVM - rbf	Yes(n_components = 107)	0.771

a. Library Imports

```
# import libraries
import os
import pandas as pd
import numpy as np

from PIL import Image
from PIL import ImageOps

import cv2
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA

# libraries for data plotting
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import seaborn as sns

# library for evaluation
from sklearn import metrics

# libraries for ML algorithms
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier

from random import randint
from random import seed
from sklearn.preprocessing import LabelEncoder
RANDOM_SEED = 100
```

b. File Imports

Folder Unzipping

```
[5] #An easier way to unzips the libraries
    #This downloads the zip file directly

    !pip install --q --upgrade --no-cache-dir gdown
    !gdown 1ea-X2EK6a49Ond-YJZ6Dt0nqCQFpW0Pi

    Downloading...
    From: https://drive.google.com/uc?id=1ea-X2EK6a49Ond-YJZ6Dt0nqCQFpW0Pi
    To: /content/data.zip
    100% 472M/472M [00:06<00:00, 67.5MB/s]
```

```
import zipfile
with zipfile.ZipFile('data.zip', 'r') as zip_ref:
    zip_ref.extractall('.')
```

c. Loading image datasets

Loading the data sets

```python
#load training data
df_train = pd.read_csv('train.csv')

#Load the test data
df_test = pd.read_csv('test.csv')


# summarise the details
print(f'Number of train entries: {len(df_train)}')
print(f'Number of test entries: {len(df_test)}')
```

```
Number of train entries: 10000
Number of test entries: 1000
```

d. Plotly Visualization code

Can Plotly tell us how many cats and dogs?

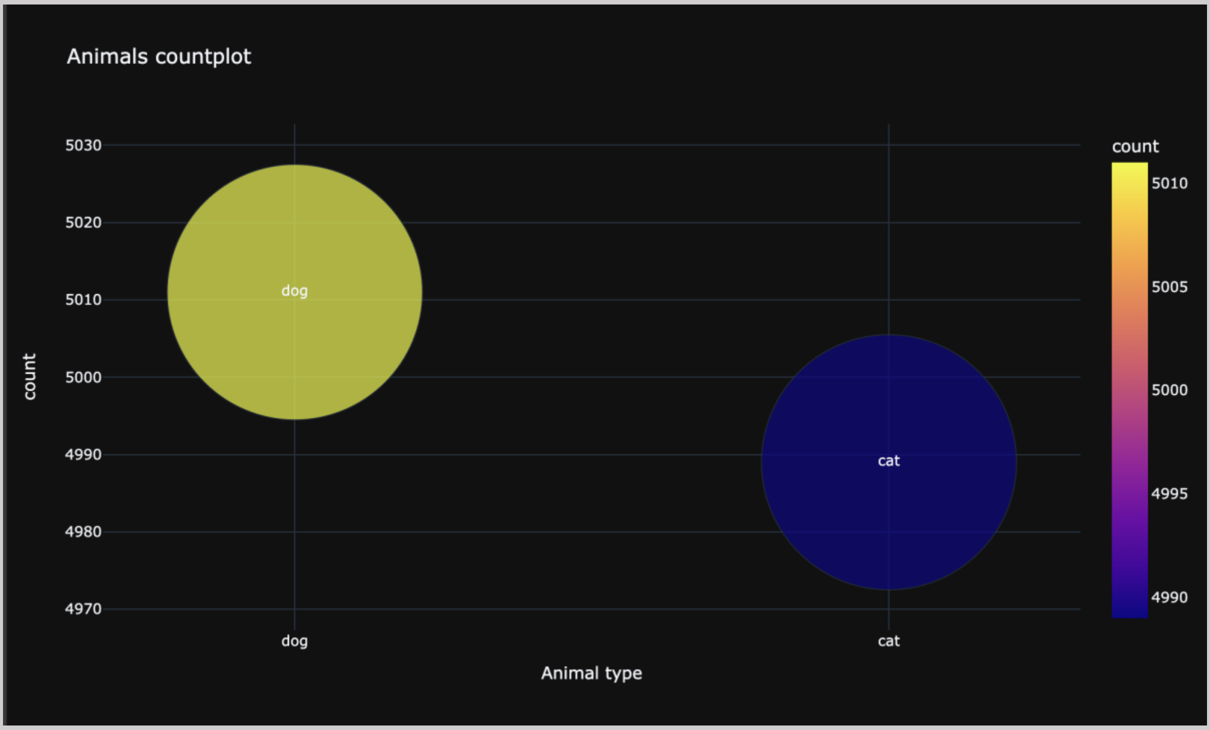
```python
counts = df_train['label'].value_counts()
counts = counts.reset_index()
counts.columns = ['animal', 'count']

fig = px.scatter(counts, x="animal", y="count",
                 size="count", size_max=150, color="count",
                 hover_name="count", text="animal",
                 title="Animals countplot")

# Set the x-axis and y-axis labels
fig.update_layout(xaxis_title="Animal type", yaxis_title="count", height=600, width=1000, template="plotly_dark")


# Show the chart
fig.show()
```

e. Visualization results



f. Loading images

Loading the test and train images

```
[12] #Create a function that loads the images and resizes if necessary, and ret
def load_images(ids, folder_path, dim):
  images = []
  for id in ids:
    image_path = os.path.join(folder_path, f'{id}.jpg')
    img = cv2.imread(image_path)

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

g. Model Evaluation

Function for Evaluation

```
# method to plot confusion matrix
def plot_confusion_matrix(matrix):
    plt.clf()
    plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Set2_r)
    classNames = ['0', '1']
    plt.title('Confusion Matrix')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FP'], ['FN', 'TP']]

    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(matrix[i][j]))
    plt.show()

# method to calculate evaluation results
def evaluate(actuals, predictions):
  accuracy = metrics.accuracy_score(actuals, predictions)
  confusion_matrix = metrics.confusion_matrix(actuals, predictions, labels=[0, 1])
  return accuracy, confusion_matrix
```

h. ML Models definition

Model building

```
[ ] #Define SVM --linear
def build_svm_lin_model(X_train, X_val, y_train, y_val):
  clf = svm.SVC(kernel='linear', random_state=RANDOM_SEED)
  clf.fit(X_train, y_train)
  return clf
```

```
#Define SVM --rbf
def build_svm_rbf_model(X_train, X_val, y_train, y_val):
  clf = svm.SVC(kernel='rbf', random_state=RANDOM_SEED)
  clf.fit(X_train, y_train)
  return clf
```

```
[ ] #Define SVM --poly
def build_svm_poly_model(X_train, X_val, y_train, y_val):
  clf = svm.SVC(kernel='poly', random_state=RANDOM_SEED)
  clf.fit(X_train, y_train)
  return clf
```

```
[ ] #Define SVM --sigmoid
def build_svm_sigm_model(X_train, X_val, y_train, y_val):
  clf = svm.SVC(kernel='sigmoid', random_state=RANDOM_SEED)
  clf.fit(X_train, y_train)
  return clf
```

```
[ ] #Define random forest
def build_rfc_model(X_train, X_val, y_train, y_val):
  clf = RandomForestClassifier(n_estimators=100, max_depth=10, random_stat
  clf.fit(X_train, y_train)
  return clf
```

i. Feature Extraction 1

Feature Extraction: grayscaling only

[17]

```python
#Create a function for grayscaling, vectorizing, and feature extract

def get_features_m1(images):
  features_list = []
  for img in images:
    #Grayscaling
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #Vectorization
    features = img_grayscaled.flatten()

    #Appending the new features to a list
    features_list.append(features)

  #Converting the list to an array
  features_list = np.array(features_list)

  #Return the list
  return features_list
```

j. Feature Extraction 2

Feature Extraction with Gaussian smoothing

```python
#Function for grayscaling, blurring, and vectorizing
def get_features_m2(images):
  features_list = []
  for img in images:
    #Grayscaling
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #Blurring the image
    img_blurred = cv2.GaussianBlur(img_grayscaled,(3,3), 2)


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

    #Append the features to a list
    features_list.append(features)

  #Convert feature list to np array
  features_list = np.array(features_list)

  #Return the array
  return features_list
```

k. Feature Extraction 3

Feature Extraction using Edge map to Vector

```python
# method to get image features
def get_features_m3(images):
  features_list = []
  for img in images:
    # image preprocessing
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # feature extraction
    edges_canny = cv2.Canny(img_grayscaled, 100, 200)
    features = edges_canny.flatten()

    features_list.append(features)

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

l. Feature Extraction 4:

Feature Extraction with HOG vectorization

```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
```

m. Feature Extraction 5:

Feature extraction with image vector and edge map to vector

```python
# method to get image features
def get_features_m5(images):
  features_list = []
  for img in images:
    # image preprocessing
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # feature extraction
    edges_canny = cv2.Canny(img_grayscaled, 100, 200)
    features1 = img_grayscaled.flatten()
    features2 = edges_canny.flatten()
    features = np.hstack((features1, features2))

    features_list.append(features)

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

n. Feature Extraction 6:

Feature Extraction with Bilateral Filtering

```python
#Function for grayscaling, blurring, and vectorizing
def get_features_m6(images):
  features_list = []
  for img in images:
    #Grayscaling
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #Blurring the image
    img_blurred = cv2.bilateralFilter(img_grayscaled, 7, 60, 60)

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

o. Feature Extraction 7:

Feature Extraction using Morphology, GaussianBlur, adaptiveThreshold

```python
def get_features_m7(images):
  features_list = []
  for img in images:
    # convert image to grayscale
    img_grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # apply Gaussian blur to reduce noise
    # apply Gaussian blur to reduce noise
    img_blurred = cv2.GaussianBlur(img_grayscaled, (3,3), 0)

    # apply adaptive thresholding to enhance contrast
    img_thresh = cv2.adaptiveThreshold(img_blurred, 280, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 11, 5)

    # apply morphology operations to remove small objects and fill gaps
    kernel = np.ones((3,3), np.uint8)
    img_morph = cv2.morphologyEx(img_thresh, cv2.MORPH_OPEN, kernel)
    img_morph = cv2.morphologyEx(img_thresh, cv2.MORPH_CLOSE, kernel)


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

    # feature extraction
    features, hog_image = hog(img_resized, orientations=9, pixels_per_cell=(8, 8),
                cells_per_block=(2, 2), block_norm='L2-Hys',transform_sqrt=True, feature_vector=True, visualize=True)

    features_list.append(features)

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

p. PCA Process

```python
# PCA
from sklearn.decomposition import PCA

# create PCA object
pca = PCA(n_components=107)

# fit PCA to features
pca.fit(features_train)

# transform features using PCA
features_train_pca = pca.transform(features_train)
```

Final Predictions using M13

Make predictions on test images using model 13

```
# feature extraction
features_test = get_features_m7(test_images)
print(features_test.shape)

pca.fit(features_test)
# transform features using PCA
features_test_pca = pca.transform(features_test)

# get model predictions
predictions = m13.predict(features_test_pca)
print(predictions)
```

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

```
[119] #save to csv
df_test.to_csv('/content/test_prediction.csv', index=False)
```

```
import json
test_file_path = "/content/test_prediction.csv"
df_test = pd.read_csv(test_file_path)
df_test = df_test[["id", "prediction"]]

data = []
for index, row in df_test.iterrows():
    data.append({'id': row['id'], 'prediction': row['prediction']})

print(data[0:5])

submission_file_path = "submission.json"
with open(submission_file_path, 'w') as fp:
    fp.write('\n'.join(json.dumps(i) for i in data))
```

```
[{'id': 1, 'prediction': 'dog'}, {'id': 2, 'prediction': 'cat'}, {'id': 3,
```