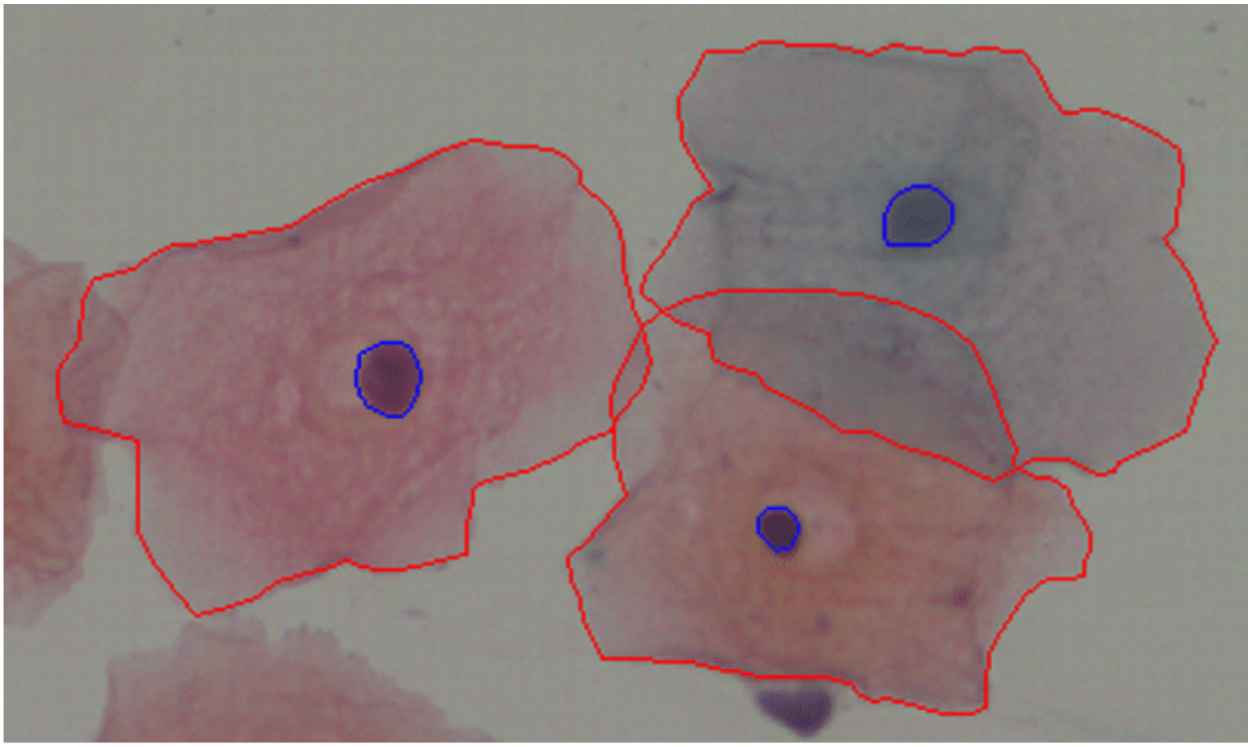
Cancer Prediction

Unsupervised learning for pap smear cancer prediction



Course : Computational Vision

Professor : Francesca Odone

Project advisor : Vito Paolo Pastore

Students :

Shayan ALvansazyazdi	5447411
Sina Hatami	5447389

Introduction

This report delves into the world of cervical cell analysis, powered by the SIPaKMeD dataset. Comprising over 4,000 meticulously curated images, this dataset is a cornerstone of our investigation into the classification of cervical cells.

Our journey begins with an exploration of the SIPaKMeD dataset and its potential applications. We leverage cutting-edge techniques, including automated feature extraction with pretrained models and traditional handcrafted feature engineering, to decode the rich information encapsulated within these cervical cell images.

Through clustering and classification methodologies, we seek to uncover patterns and assess the accuracy of our models in distinguishing between cervical cell categories. This report provides a comprehensive overview of our findings, offering valuable insights into the world of cervical health analysis.

By the end of this report, you will gain a deeper understanding of how the SIPaKMeD dataset can be harnessed for medical image analysis, showcasing the advantages and limitations of different feature extraction approaches.

Data Description

The SIPaKMeD dataset, publicly available for experimental use, constitutes a pivotal resource for the analysis and classification of cervical cells. This dataset was meticulously assembled and made accessible through the work of Marina E. Plissiti et al., as presented in their paper titled "SIPAKMED: A new dataset for feature and image-based classification of normal and pathological cervical cells in Pap smear images" at the IEEE International Conference on Image Processing (ICIP) in 2018.

Dataset Composition

The SIPaKMeD Database encompasses a total of 4,049 images, each of which has been carefully isolated from 966 cluster cell images sourced from Pap smear slides. These images were acquired via a CCD camera adapted to an optical microscope, ensuring the acquisition of high-quality representations of cervical cells. The dataset is methodically organized into five distinct categories, each with its own significance in the realm of cervical health assessment:

- **Superficial-Intermediate Cells** : These cells represent the majority of the cells encountered in Pap tests. They are typically characterized by their flat or polygonal shapes, eosinophilic or cyanophilic cytoplasm staining, central pycnotic nuclei, and well-defined nuclear boundaries. Notably, these cells exhibit morphological changes, including koilocytic atypia, in cases of more severe lesions.

- **Parabasal Cells** : Parabasal cells, the smallest epithelial cells observed in vaginal smears, are considered immature squamous cells. They exhibit cyanophilic cytoplasm and generally possess large vesicular nuclei. It's worth noting that parabasal cells share morphological characteristics with metaplastic cells, making them challenging to distinguish.

- **Koilocytotic Cells** : Koilocytotic cells are primarily found in mature squamous cells, including intermediate and superficial types. They tend to appear cyanophilic, lightly stained, and feature a large perinuclear cavity. The nuclei of koilocytes are typically enlarged, eccentrically located, hyperchromatic, and exhibit irregular nuclear membrane contours.

- **Dyskeratotic Cells** : Dyskeratotic cells are squamous cells that have prematurely undergone abnormal keratinization, either individually or more frequently within three-dimensional clusters. These cells exhibit an orangeophilic cytoplasm and

vesicular nuclei, resembling koilocytes. In many instances, they present as binucleated and/or multinucleated cells.

- **Metaplastic Cells** : Metaplastic cells, in essence, resemble small or large parabasal-type cells, featuring prominent cellular borders, eccentric nuclei, and occasional large intracellular vacuoles. They are characterized by lighter brown staining in the central portion and darker-stained cytoplasm in the marginal portion. Metaplastic cells exhibit a remarkable uniformity in size and shape compared to parabasal cells, with well-defined, nearly round cytoplasm.

This dataset, with its comprehensive coverage of cervical cell variations, offers a pivotal resource for advancing the development of automated diagnostic tools and image-based classification algorithms within the field of cervical health assessment. Researchers, clinicians, and data scientists can leverage this dataset to enhance our understanding of cervical cell morphology and contribute to the improved early detection of abnormalities in Pap smear images.

Unsupervised Clustering Using VGG16 Features

```
import os
import numpy as np
import cv2
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Set the path to the dataset main folder
dataset_path = '/content/drive/MyDrive/Computer_vision'

# Define functions for data preprocessing and feature extraction

def preprocess_images(image_paths, target_size=(224, 224)):
    images = []
    for image_path in image_paths:
        img = cv2.imread(image_path)
        img = cv2.resize(img, target_size)
        images.append(img)
    return np.array(images)

# Load and preprocess the dataset
image_paths = []  # Store paths to image files
labels = []       # Store labels for each image
label_encoder = LabelEncoder()  # Create a label encoder

for folder_name in os.listdir(dataset_path):
    folder_path = os.path.join(dataset_path, folder_name)
    if os.path.isdir(folder_path):
        for file_name in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file_name)
            if file_name.endswith('.bmp'):
                image_paths.append(file_path)
                labels.append(folder_name)  # Use folder names as labels

# Encode labels using the label encoder
labels = label_encoder.fit_transform(labels)

images = preprocess_images(image_paths)
```

This code block initializes essential functionalities for data preprocessing and feature extraction. It defines functions to preprocess images by resizing them to a specified target size and sets up the necessary structures to organize image paths, labels, and a label encoder. Additionally, it loads and preprocesses the dataset by iterating through the dataset folders, extracting image paths and labels for further use.

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Load pre-trained VGG16 model
base_model = VGG16(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('fc2').output)

# Preprocess images for VGG16
preprocessed_images = preprocess_input(images)

# Extract features
pretrained_features = model.predict(preprocessed_images)

# Perform clustering
num_samples, num_features = pretrained_features.shape
pretrained_features_2d = pretrained_features.reshape(num_samples, -1)
num_clusters = 5  # Specify the number of clusters
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
clusters = kmeans.fit_predict(pretrained_features)

# Evaluate clustering
accuracy = accuracy_score(labels, clusters)
precision = precision_score(labels, clusters, average='macro')
recall = recall_score(labels, clusters, average='macro')
f1 = f1_score(labels, clusters, average='macro')
print(f'Clustering Accuracy: {accuracy:.2f}')
print(f'Clustering Precision: {precision:.2f}')
print(f'Clustering Recall: {recall:.2f}')
print(f'Clustering F1-score: {f1:.2f}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
553467096/553467096 [==============================] - 6s 0us/step
31/31 [==============================] - 561s 18s/step
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning
  warnings.warn(
Clustering Accuracy: 0.26
Clustering Precision: 0.25
Clustering Recall: 0.30
Clustering F1-score: 0.26
```

In this code block, we first load a pre-trained VGG16 model, and then preprocess the images using VGG16's specific preprocessing function. Next, we extract features from the preprocessed images using the VGG16 model up to the 'fc2' layer.

We proceed to perform clustering using KMeans on the extracted features, aiming to group similar features into specified clusters (in this case, 5 clusters). The clustering results are evaluated using clustering metrics such as accuracy, precision, recall, and F1-score.

The accuracy indicates the proportion of correctly clustered samples, while precision measures the fraction of true positives among all samples assigned to a specific cluster. Recall assesses the proportion of true positives among all actual instances of a particular class, and F1-score is the harmonic mean of precision and recall, offering a balanced evaluation of clustering performance.

The clustering results using VGG16 features indicate a relatively poor performance with an accuracy of 26%. The low precision and F1-score further suggest that the clusters may not accurately represent the underlying patterns in the data. Improvements in clustering methods or feature representations may be beneficial for more meaningful grouping of the images.

Classification using Handcrafted Features and SVM

```python
from skimage.feature import hog

# Extract handcrafted features using HOG
def extract_handcrafted_features(images, multichannel=True):
    hog_features = []
    for image in images:
        hog_feature = hog(image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), multichannel=multichannel)
        hog_features.append(hog_feature)
    return np.array(hog_features)

handcrafted_features = extract_handcrafted_features(images, multichannel=True)
```

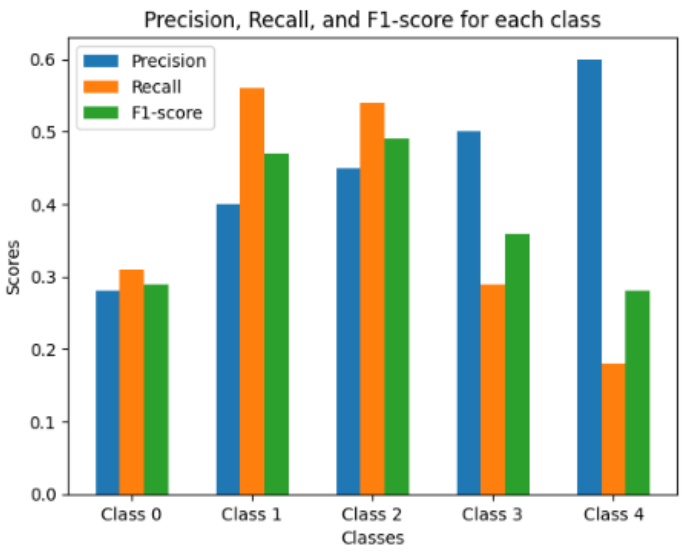
```
<ipython-input-27-381ac4454b81>:7: FutureWarning: `multichannel` is a deprecated argument name for `hog`. I
  hog_feature = hog(image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), multichannel=multichannel)
```

This part of the code extracts handcrafted features using Histogram of Oriented Gradients (HOG) from the preprocessed images. The HOG features are computed

with specific parameters like pixels per cell and cells per block. The features are then stored in an array named “handcrafted_features”.

```python
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Split the dataset for classification
X_train, X_test, y_train, y_test = train_test_split(handcrafted_features, labels, test_size=0.2, random_state=42)

# Train a classifier (e.g., SVM) using the extracted features
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)

# Predict labels
y_pred = clf.predict(X_test)

# Evaluate classifier performance
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Class	Precision	Recall	F1-Score
Class 0	0.28	0.31	0.29
Class 1	0.40	0.56	0.47
Class 2	0.45	0.54	0.49
Class 3	0.50	0.29	0.36
Class 4	0.60	0.18	0.28



This part of the code trains a Support Vector Machine (SVM) classifier using the handcrafted features and evaluates its performance using metrics like precision, recall, and F1-score. The output provides a classification report showing these metrics for each class and an overall summary.

In comparison to the previous clustering approach, this classification approach provides a more detailed evaluation of the model's performance on individual classes, yielding higher precision, recall, and F1-score. However, the accuracy is still relatively low, suggesting that further improvements may be needed.

Comparing Classification Accuracy :

Pretrained vs. Handcrafted Features

```
[30] # Split the dataset for classification
     X_train_handcrafted, X_test_handcrafted, y_train, y_test = train_test_split(handcrafted_features, labels, test_size=0.2, random_state=42)

     # Calculate accuracy for handcrafted features
     clf_handcrafted = SVC(kernel='linear')
     clf_handcrafted.fit(X_train_handcrafted, y_train)
     y_pred_handcrafted = clf_handcrafted.predict(X_test_handcrafted)
     accuracy_handcrafted = accuracy_score(y_test, y_pred_handcrafted)
```

```
# Split the dataset into training and testing sets for pretrained features
X_train_pretrained, X_test_pretrained, y_train, y_test = train_test_split(pretrained_features_2d, labels, test_size=0.2, random_state=42)

# Train a classifier using pretrained features
clf_pretrained = SVC(kernel='linear')
clf_pretrained.fit(X_train_pretrained, y_train)

# Predict labels using pretrained features
y_pred_pretrained = clf_pretrained.predict(X_test_pretrained)

# Calculate accuracy for pretrained features
accuracy_pretrained = accuracy_score(y_test, y_pred_pretrained)


# Print the accuracies
print(f'Accuracy (Pretrained Features): {accuracy_pretrained:.2f}')
print(f'Accuracy (Handcrafted Features): {accuracy_handcrafted:.2f}')
```

```
Accuracy (Pretrained Features): 0.85
Accuracy (Handcrafted Features): 0.40
```

In this part, we split the dataset for classification into training and testing sets for both handcrafted and pretrained features. We then trained SVM classifiers on these features and calculated the accuracies.

- Accuracy for Pretrained Features: 0.85

- Accuracy for Handcrafted Features: 0.40

The accuracy for pretrained features (VGG16) is significantly higher than that for handcrafted features (HOG), indicating that the pretrained CNN features perform much better in this classification task.

In addition the accuracy for pre-trained features (0.85) is notably higher than the accuracy achieved through clustering (0.26) and SVM with HOG-based handcrafted features (0.40). This indicates that using pre-trained features from VGG16 for classification produces superior results compared to the clustering approach and handcrafted features with SVM.

Robustness and Performance:

K-Fold Cross-validation with Pretrained Features and SVM

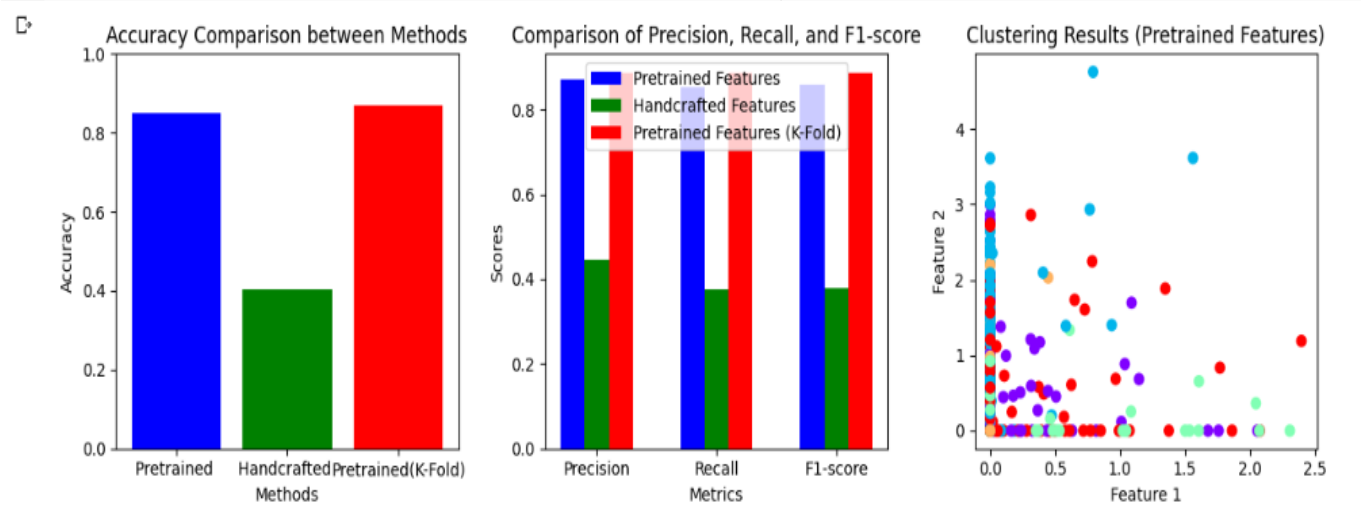
```python
from sklearn.model_selection import StratifiedKFold

# Define the number of folds for k-fold cross-validation
num_folds = 5

# Initialize StratifiedKFold
kf = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)

# Lists to store evaluation results
accuracy_values_kfold = []
precision_values_kfold = []
recall_values_kfold = []
f1_values_kfold = []

# Perform k-fold cross-validation
for train_index, test_index in kf.split(pretrained_features_2d, labels):
    X_train_kfold, X_test_kfold = pretrained_features_2d[train_index], pretrained_features_2d[test_index]
    y_train_kfold, y_test_kfold = labels[train_index], labels[test_index]

    # Train a classifier using pretrained features
    clf_pretrained_kfold = SVC(kernel='linear')
    clf_pretrained_kfold.fit(X_train_kfold, y_train_kfold)

    # Predict labels using pretrained features
    y_pred_pretrained_kfold = clf_pretrained_kfold.predict(X_test_kfold)

    # Calculate evaluation metrics for pretrained features
    accuracy_kfold = accuracy_score(y_test_kfold, y_pred_pretrained_kfold)
    precision_kfold = precision_score(y_test_kfold, y_pred_pretrained_kfold, average='macro')
    recall_kfold = recall_score(y_test_kfold, y_pred_pretrained_kfold, average='macro')
    f1_kfold = f1_score(y_test_kfold, y_pred_pretrained_kfold, average='macro')

    # Append the evaluation results for this fold
    accuracy_values_kfold.append(accuracy_kfold)
    precision_values_kfold.append(precision_kfold)
    recall_values_kfold.append(recall_kfold)
    f1_values_kfold.append(f1_kfold)

# Print the aggregated results for k-fold cross-validation
print("K-Fold Cross-validation results:")
print(f"Mean Accuracy: {np.mean(accuracy_values_kfold):.2f}")
print(f"Mean Precision: {np.mean(precision_values_kfold):.2f}")
print(f"Mean Recall: {np.mean(recall_values_kfold):.2f}")
print(f"Mean F1-score: {np.mean(f1_values_kfold):.2f}")
```

```
K-Fold Cross-validation results:
Mean Accuracy: 0.87
Mean Precision: 0.89
Mean Recall: 0.89
Mean F1-score: 0.89
```

This part of the code performs k-fold cross-validation using pretrained features with a linear SVM classifier. The dataset is split into 5 folds, and the classifier is trained and evaluated on each fold separately. The evaluation metrics (accuracy, precision, recall, and F1-score) are then averaged over the folds to provide an overall assessment of the model's performance.

The k-fold cross-validation results indicate high performance, with an average accuracy, precision, recall, and F1-score of approximately 0.87-0.89, suggesting the robustness and effectiveness of the model in classifying the dataset.

Final Visual Comparing



We can see the ame result of different methods all together and as we already know K-fold obtained the best result while pre-trained features performance was close to it.

Confusion Matrix

A confusion matrix is a table often used to describe the performance of a classification model (in this case, it's used for evaluating the performance of clustering as well). It allows visualization of the true positive, true negative, false positive, and false negative values for a set of data points.

The confusion matrix provides a clear representation of how well the clustering algorithm performed in assigning data points to the correct clusters