**19CSE432**

**Pattern Recognition**

**Case Study Review**

**Title: Object Recognition**

**Team Members:**

1. Aadithya E R(CB.EN.U4CSE21301)

Implemented object recognition, Gaussian Naïve Bayes Classifier and Dimensionality reduction(PCA)

1. Parthan J (CB.EN.U4CSE21321)

Implemented Bayesian Belief Network, DBSCAN.

**Object recognition**:

Object recognition is a computer vision technique that involves identifying and detecting objects within digital images or videos.

The process of object recognition typically involves several steps:

1. Image acquisition: collection of visual data
2. Preprocessing: Enhancing or modifying the image to improve the quality and make it suitable for analysis.
3. Feature Extraction: Identifying distinctive features within the image that are relevant for recognizing objects.
4. Object detection: Locating and identifying specific objects within the image. This step involves using algorithms that can detect objects based on learned patterns and features extracted from the image.
5. Classification and Recognition: Assigning labels or categories to the detected objects based on their characteristics.

**Pattern Recognition concepts used in the case study:**

1. Image Classification with Pre-trained Models (MobileNetV2):

* Uses a pre-trained MobileNetV2 model from Keras to perform image classification on a given image.
* Utilizes the model to predict and visualize the top predictions for objects present in the image.

1. Dimensionality Reduction (PCA) in Object Recognition:

* Employs Principal Component Analysis (PCA) to reduce the dimensionality of features for Forest Cover type predeiction.
* Visualizes the reduced-dimensional representations of features obtained from PCA.

1. Visualizing Intermediate Features from a Convolutional Neural Network (MobileNetV2):

* Extracts intermediate features from a specific layer ("block\_1\_expand\_relu") of the MobileNetV2 model.
* Displays the intermediate features as images, showing how the network detects different patterns or features at an intermediate level.

1. Bayesian Belief Network (Bayesian Network) Implementation:

* Utilizes the ‘pgmpy’ library to create a Bayesian Network representing relationships between features and objects.
* Checks the model's consistency and performs inference to calculate probabilities based on given evidence in the network.

1. Implementation of DBSCAN:

* Utilizes the ‘DBSCAN’ library to create a DBSCAN representing relationships between features and objects.

**Programming language used:**

Python

**Python Libraries used:**

TensorFlow, Keras, Numpy, Matplotlib, pgmpy

**Dataset Description:**

The dataset used for the code execution contains images of various objects across different classes. The size of the image should be of the resolution 224\*224 pixels.

**Challenges:**

1. **Model Generalization:** The pre-trained MobileNetV2 model was trained on the ImageNet dataset, which comprises a large and diverse set of images across multiple classes.
2. **Accuracy and Confidence:** the confidence scores associated with these predictions might not always reflect the actual accuracy. Some objects might be misclassified or have low confidence scores due to factors like image quality, background clutter, or object occlusion.
3. **Handling Multiple Objects:** The model might detect multiple objects in the image. Visualizing and interpreting multiple predictions could be complex, especially when objects overlap or are present in cluttered backgrounds.
4. **Performance on Specific Objects:** For some specific objects, the model might face challenges in correctly identifying them due to similarities with other objects, varying scales, orientations, or context within the image

**Code explanation:**

**Object recognition**

**Import Libraries:** The script imports necessary libraries such as TensorFlow, Keras, NumPy, and Matplotlib.

**Load Pre-trained MobileNetV2 Model:** The script loads the MobileNetV2 model pre-trained on the ImageNet dataset.

**Preprocess Image Function:**

**preprocess\_image:** This function takes an image path as input, loads the image, resizes it to (224, 224) pixels (MobileNetV2 input size), converts it into a NumPy array, and preprocesses it using preprocess\_input from Keras.

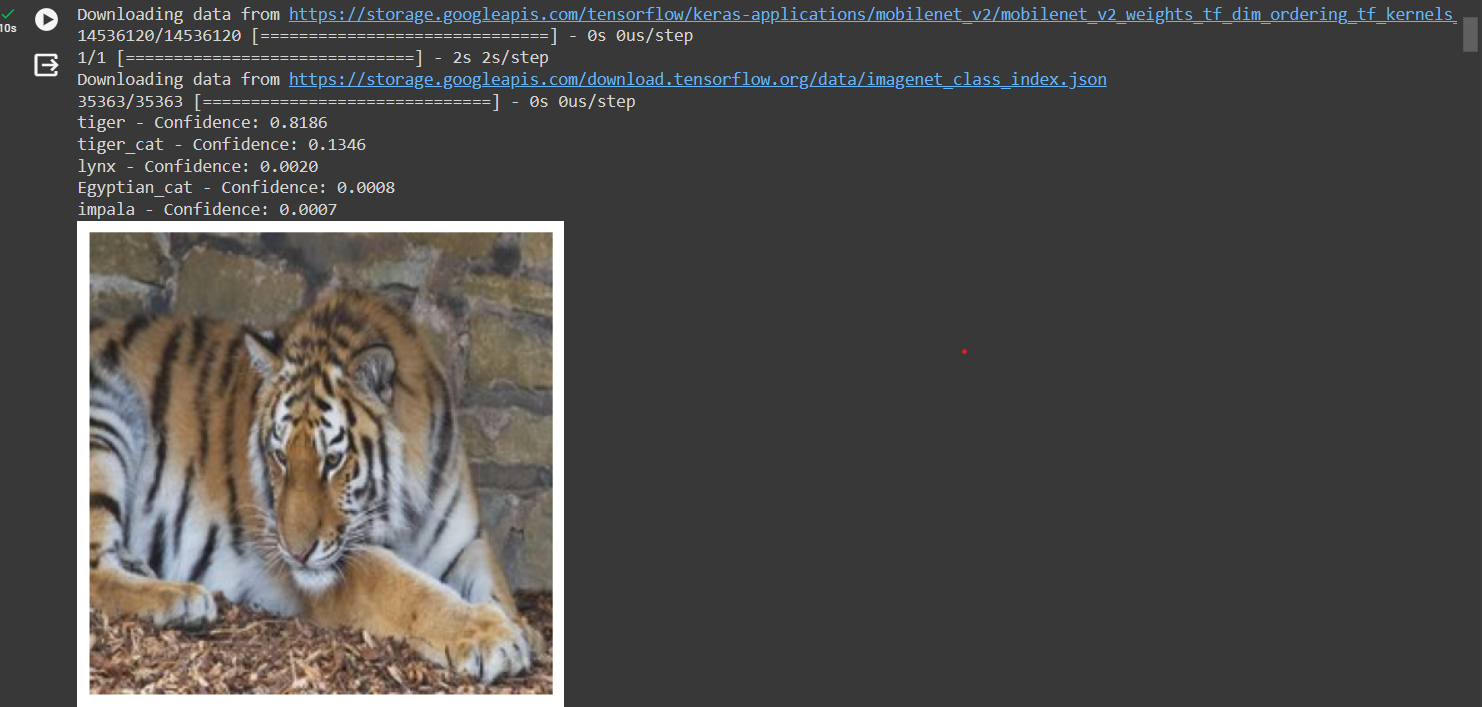
**Prediction and Visualization Functions:**

**predict\_and\_visualize:** This function performs prediction using the pre-trained MobileNetV2 model on the input image. It preprocesses the image, predicts its class probabilities using model.predict, decodes the predictions to obtain the top 5 classes with their confidence scores using decode\_predictions, and displays the image along with a horizontal bar chart showing the top predictions and their confidence scores.

**Execution:**

The script specifies the path to the input image (img\_path) and calls the predict\_and\_visualize function to perform image classification and visualization on the provided image.

**Output:**



A screenshot of a computer screen

Description automatically generated

**Code explanation:**

**Intermediate features:**

**Model Loading:** The code begins by loading the pre-trained MobileNetV2 model using TensorFlow's Keras API. This model is pre-trained on the ImageNet dataset and serves as a feature extractor.

**Preprocessing Function:** The preprocess\_image function takes the path to an input image and preprocesses it to meet the model's input requirements by resizing and normalizing the image.

**Feature Extraction Function:** The extract\_intermediate\_features function takes an input image's path and a layer name as arguments. It loads the image, extracts intermediate features from the specified layer of the model using a functional Keras model, and returns the output of that layer for the provided image.

**Image Path:** The img\_path variable holds the path to the input image ('tiger (1).jpg' in this case).

**Visualization of Intermediate Features:** The code segment visualizes a selection of intermediate features extracted from the specified layer of the MobileNetV2 model. It plots 9 features as subplots in a 3x3 grid, using the imshow function from Matplotlib to display each feature's content and the cmap='viridis' parameter to assign a colormap for visualization.

**Display:** Finally, plt.show() displays the visualized intermediate features in a grid layout, with a title indicating that these are intermediate features extracted from the MobileNetV2 model.

A screenshot of a computer

Description automatically generated

**Code explanation**

**Bayesian Belief network**

**Bayesian Network Creation**: The code initializes an empty Bayesian Network (model) with nodes 'Feature1', 'Feature2', and 'Object', where 'Feature1' and 'Feature2' are the parent nodes influencing the 'Object' node.

**Conditional Probability Distributions (CPDs):** Conditional probability distributions for each node are defined using TabularCPD. These CPDs specify the probabilities of each state of the variable given the states of its parents.

**Adding CPDs to the Network:** The defined CPDs are added to the model using add\_cpds().

**Model Consistency Check:** model.check\_model() is used to verify if the model structure and CPDs are consistent with each other.

**Inference using Variable Elimination**: An inference object (inference) is created using Variable Elimination, allowing for probabilistic queries on the network. The query() function is used to compute the probability distribution of the 'Object' node given evidence that 'Feature1' is in state 1 and 'Feature2' is in state 0.

**Displaying Query Results:** The code prints the probability distribution for 'Object' given the specified evidence.

**Gaussian Belief Network:**

* Gaussian Naïve Bayes Classifier model to predict whether a person makes over 50K a year. The model yields a very good performance as indicated by the model accuracy which was found to be 0.8083.
* The training-set accuracy score is 0.8067 while the test-set accuracy to be 0.8083. These two values are quite comparable. So, there is no sign of overfitting. the model accuracy score which is 0.8083 with null accuracy score which is 0.7582. So, we can conclude that our Gaussian Naïve Bayes classifier model is doing a very good job in predicting the class labels.
* ROC AUC of our model approaches towards 1. So, we can conclude that our classifier does a very good job in predicting whether a person makes over 50K a year. Using the mean cross-validation, we can conclude that we expect the model to be around 80.63% accurate on average.
* Using 10 cross fold validation, we can also conclude that there is a relatively small variance in the accuracy between folds, ranging from 81.35% accuracy to 79.64% accuracy.
* So, we can conclude that the model is independent of the particular folds used for training.Our original model accuracy is 0.8083, but the mean cross-validation accuracy is 0.8063. So, the 10-fold cross-validation accuracy does not result in performance improvement for this model.

**A screen shot of a computer program

Description automatically generated**

**Null accuracy:**

**A screenshot of a computer

Description automatically generated**

**Confusion matrix:**

**A screenshot of a computer program

Description automatically generated**

**Heatmap:**

**A screenshot of a computer

Description automatically generated**

**Classification metrics:**

**A screenshot of a computer

Description automatically generated**

**Histogram plotting:**

**A screenshot of a computer

Description automatically generated**

**ROC curve:**

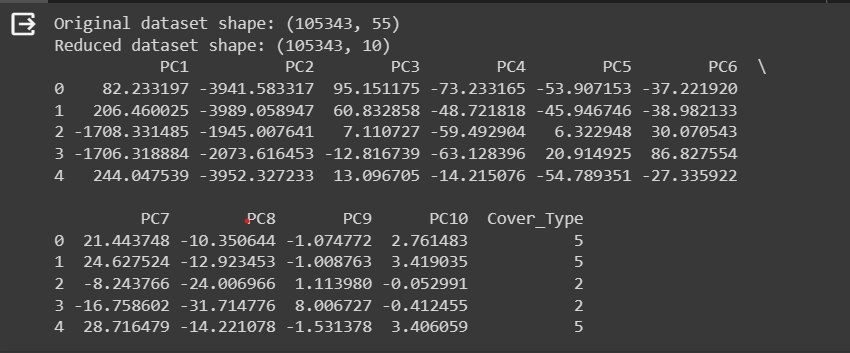
**A screen shot of a computer

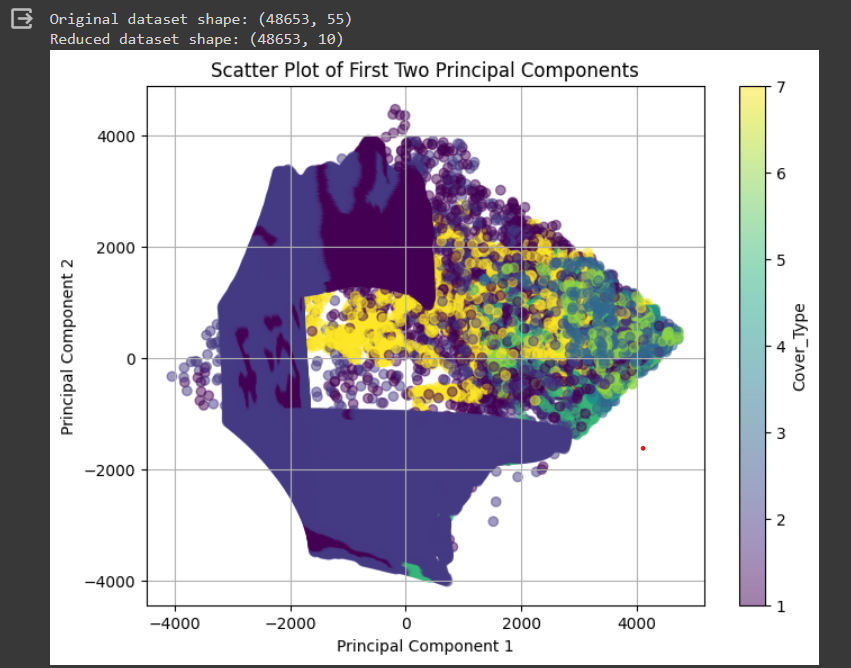
Description automatically generated**

**Dimensionality Reduction:**

* Loads the Covertype dataset and converts it into a pandas DataFrame.
* Uses SimpleImputer to handle missing values by replacing them with the mean of each feature.
* Applies PCA with n\_components=10 to reduce the dataset to 10 principal components.
* Concatenates the reduced dataset with target labels ('Cover\_Type') for further analysis or predictions.
* Displays the first few rows of the final DataFrame.

Output:





**DBSCAN and Bayesian Belief Network:**

1. Importing Libraries:

* Importing necessary libraries such as numpy, pandas, matplotlib.pyplot, seaborn, DBSCAN, StandardScaler from sklearn, and modules from pgmpy for working with Bayesian Networks.

1. Loading the Dataset:

* Loading a dataset named 'adult.csv' into a Pandas DataFrame named data.
* DBSCAN Clustering:
* Defining the features ('age' and 'hours\_per\_week') on which DBSCAN clustering will be performed.
* Standardizing the features using StandardScaler.
* Applying DBSCAN clustering using DBSCAN from sklearn.cluster.
* Adding the cluster labels generated by DBSCAN to the DataFrame.

**Visualizing DBSCAN Clusters:**

* Creating a scatter plot to visualize the clusters formed by DBSCAN using Seaborn's sns.scatterplot.
* The scatter plot shows the distribution of points in the space defined by the 'age' and 'hours\_per\_week' features, colored according to the clusters identified by DBSCAN.

**Bayesian Belief Network (BBN):**

Defining the structure of a Bayesian Network using BayesianNetwork from pgmpy.models.

Defining Conditional Probability Distributions (CPDs) for the nodes 'Age', 'HoursPerWeek', and 'Income'.

Adding the CPDs to the Bayesian network using the add\_cpds method.

**Model Consistency Check:**

* Checking the consistency of the Bayesian Network model using check\_model().
* Variable Elimination and Querying:
* Creating an inference object using VariableElimination for the Bayesian Network model.
* Performing a query to find the probability distribution for 'Income' given 'Age' = 1 and 'HoursPerWeek' = 0.

**Code Explanation:**

The code performs DBSCAN clustering on the 'age' and 'hours\_per\_week' features from the loaded dataset, visualizes the resulting clusters, constructs a Bayesian Network with specified CPDs, and queries the probability distribution in the Bayesian Network based on the provided evidence.