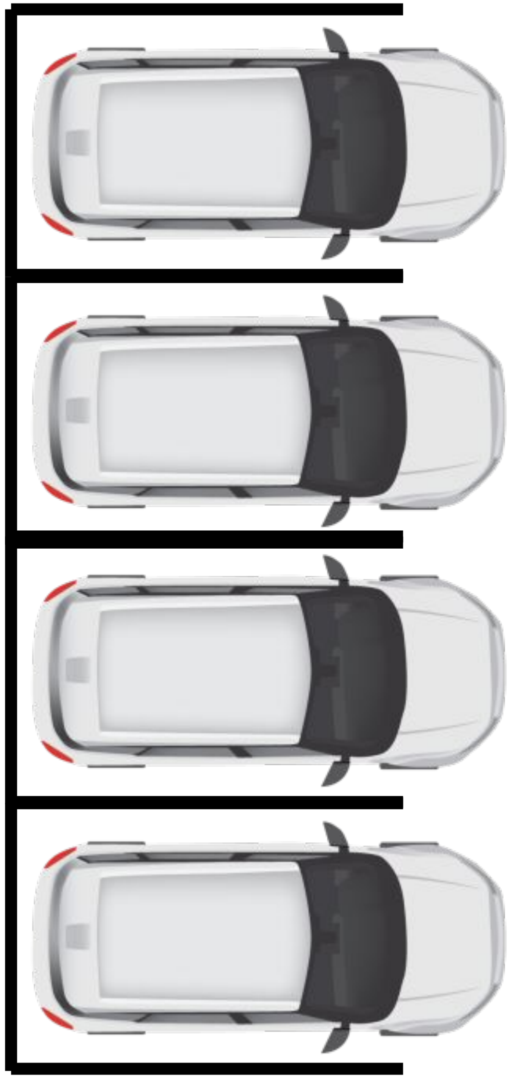
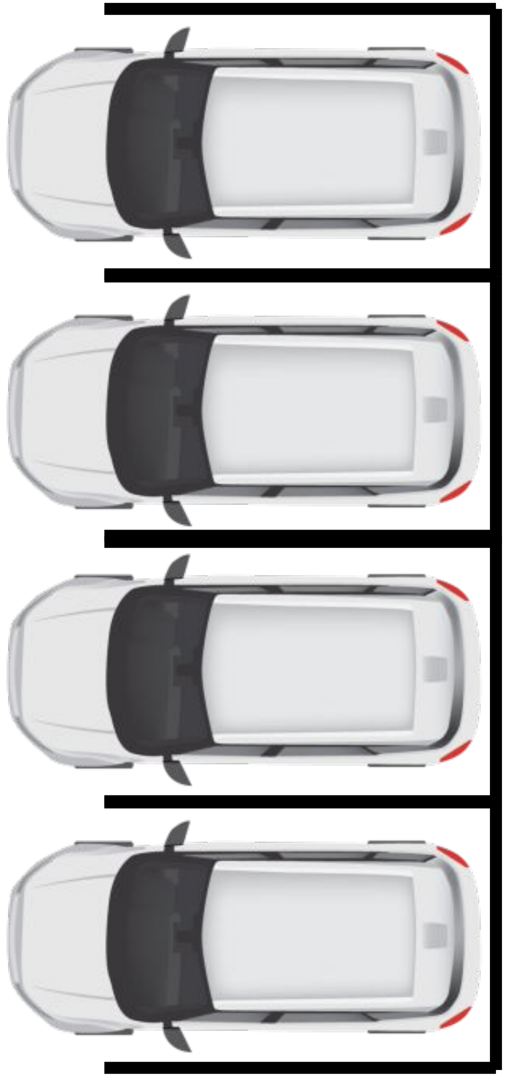
Parking Occupancy Detection





Presented by

Dariana Dorin
Djibril Coulybaly

What we will cover today

1

Introduction

Outlining the objective of project

2

Components

Needed to complete the project

3

Models

State-of-the-Art to Basic Classification Algorithms

4

Implementation

What we worked on throughout the project

5

Evaluation

Comparing the results of our models/C.A prediction

6

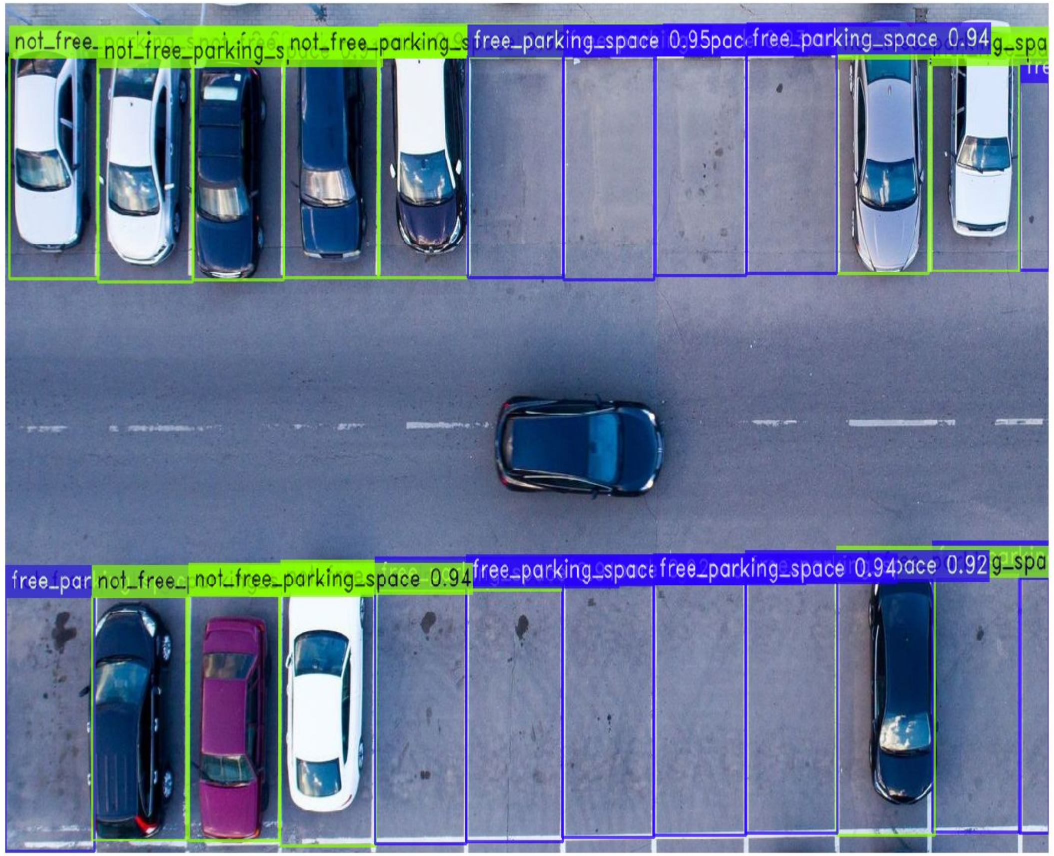
Conclusion

What we've learned and future improvements

1

Introduction

Introduction of Project



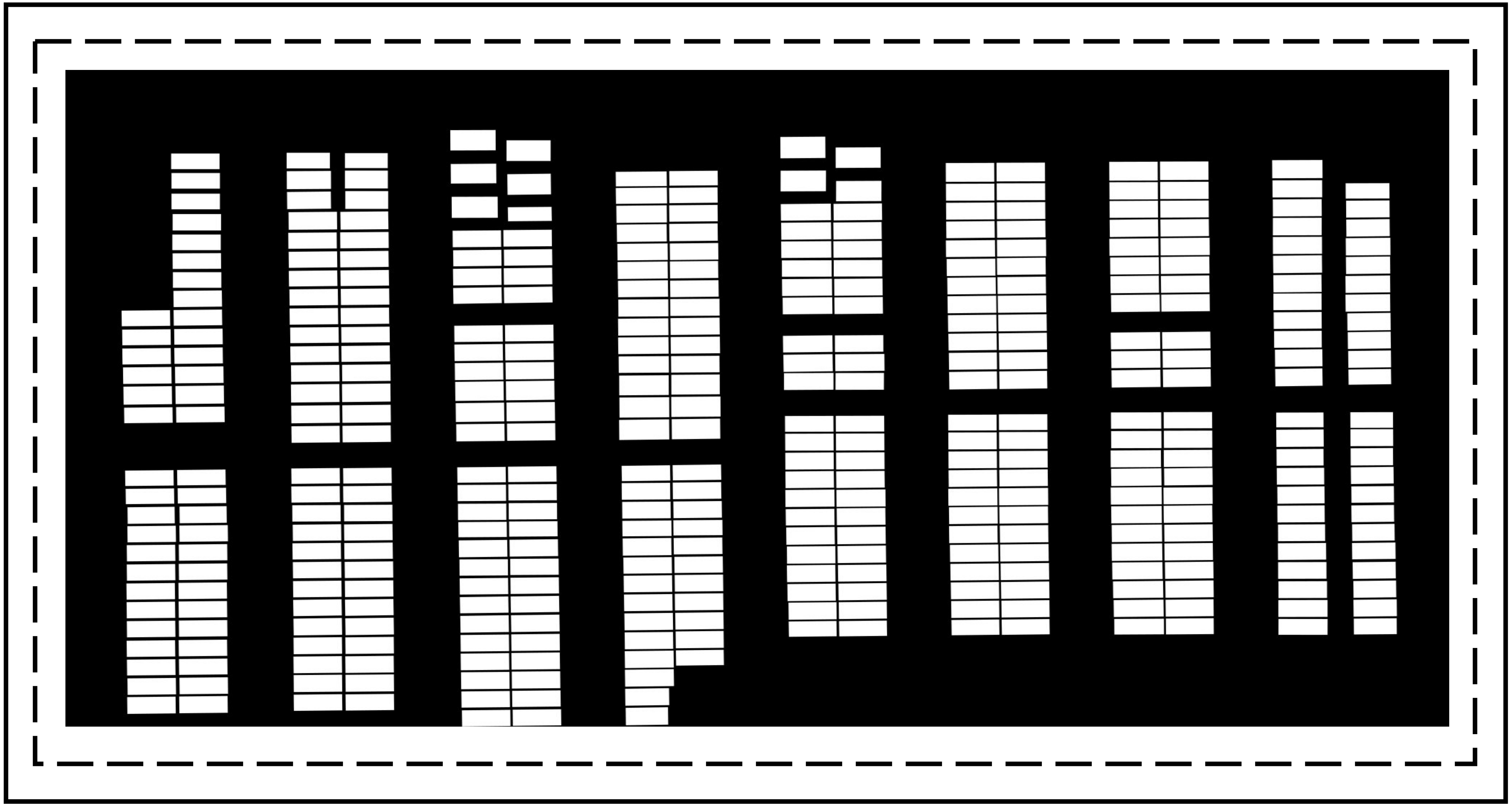
- Detect whether parking spot is occupied by a vehicle/object or not
- Segment parking spaces from video data
- Real-time monitoring/data on occupancy count
- Can be applied to parking spaces in shopping centres, garages, airports etc.

2

Components

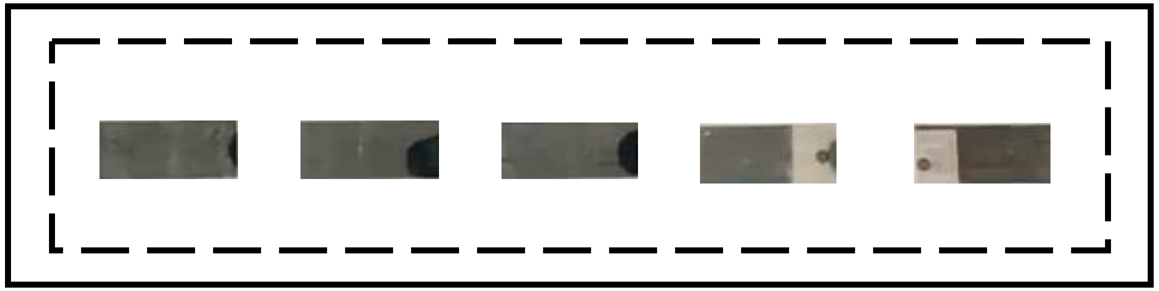
Video of Car Park

Mask of Car Park

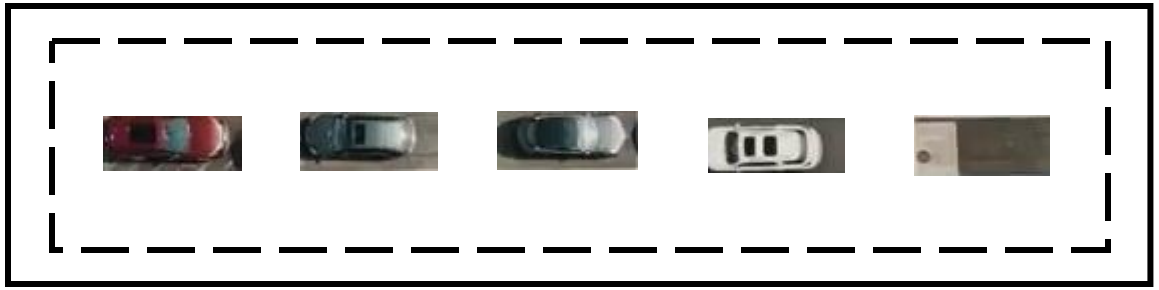


Dataset

Empty



Not Empty



Technology Used

OpenCV

OpenCV

Image/Video Processing

TensorFlow

TensorFlow

Building, training and testing deep learning models

MatPlotLib

Visual Representation of Data

Technology Used

SKLearn

Evaluating model performance

Numpy

Handling numerical operations

Python

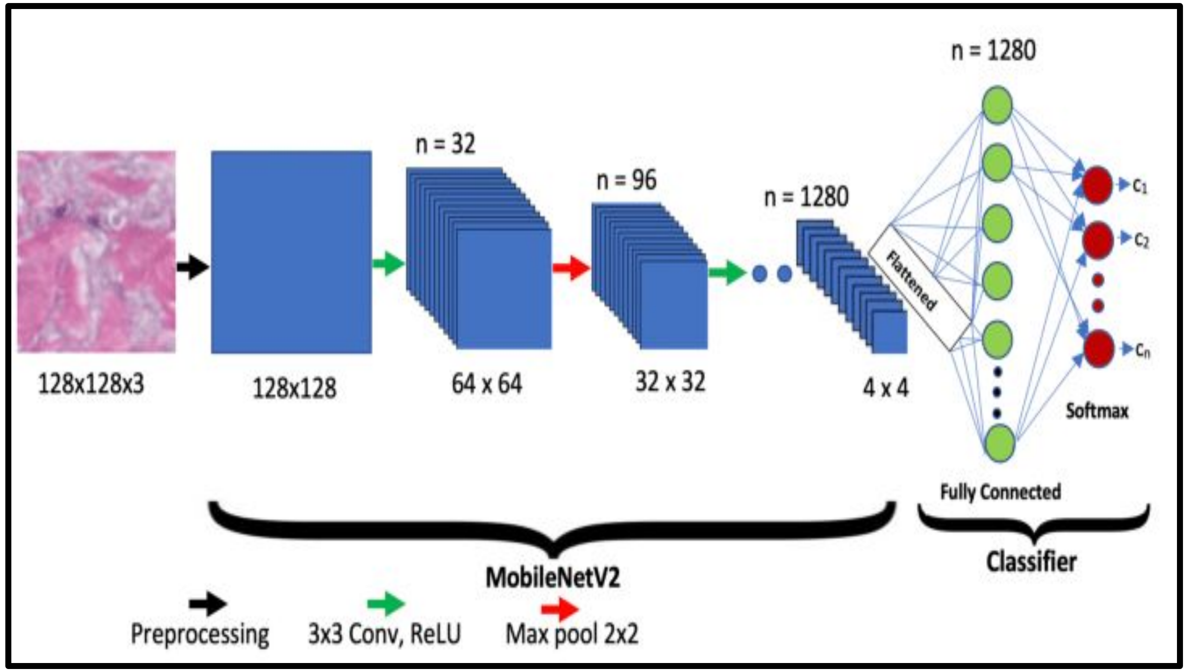
Programming language to complete project

3

Models

MobileNetV2

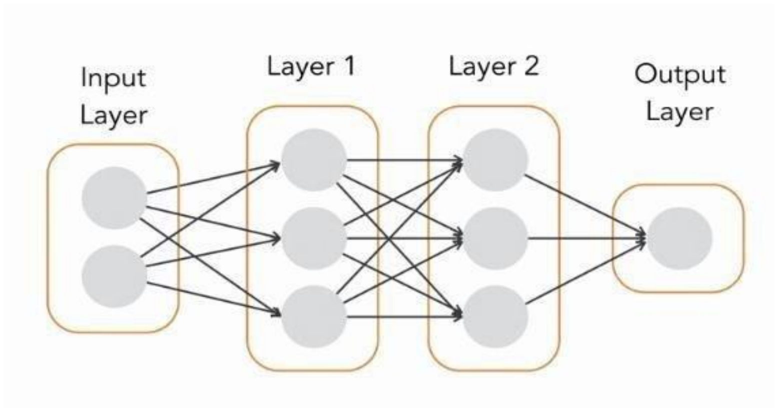
- Recommended image target size = 224x224 (not optimal for our data set)
- Image size = 32x75 (min 32 required)



MobileNetV2

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 6s 28ms/step - accuracy: 0.9045 - loss: 0.2554 - val_accuracy: 0.9458 - val_loss: 0.1669
Epoch 2/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 26ms/step - accuracy: 0.9952 - loss: 0.0286 - val_accuracy: 0.9606 - val_loss: 0.1474
Epoch 3/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.9971 - loss: 0.0166 - val_accuracy: 0.9680 - val_loss: 0.1155
Epoch 4/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 0.9997 - loss: 0.0095 - val_accuracy: 0.9672 - val_loss: 0.1036
Epoch 5/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.9999 - loss: 0.0061 - val_accuracy: 0.9721 - val_loss: 0.0968
Epoch 6/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 1.0000 - loss: 0.0050 - val_accuracy: 0.9713 - val_loss: 0.0979
Epoch 7/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 1.0000 - loss: 0.0034 - val_accuracy: 0.9672 - val_loss: 0.0966
Epoch 8/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.9680 - val_loss: 0.0969
Epoch 9/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 1.0000 - loss: 0.0025 - val_accuracy: 0.9680 - val_loss: 0.0936
Epoch 10/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.9680 - val_loss: 0.0938
39/39 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9730 - loss: 0.0726
Validation Loss: 0.0938146561384201
Validation Accuracy: 0.9679803252220154
```

Keras Sequential Model



Keras

Image size = 29x68

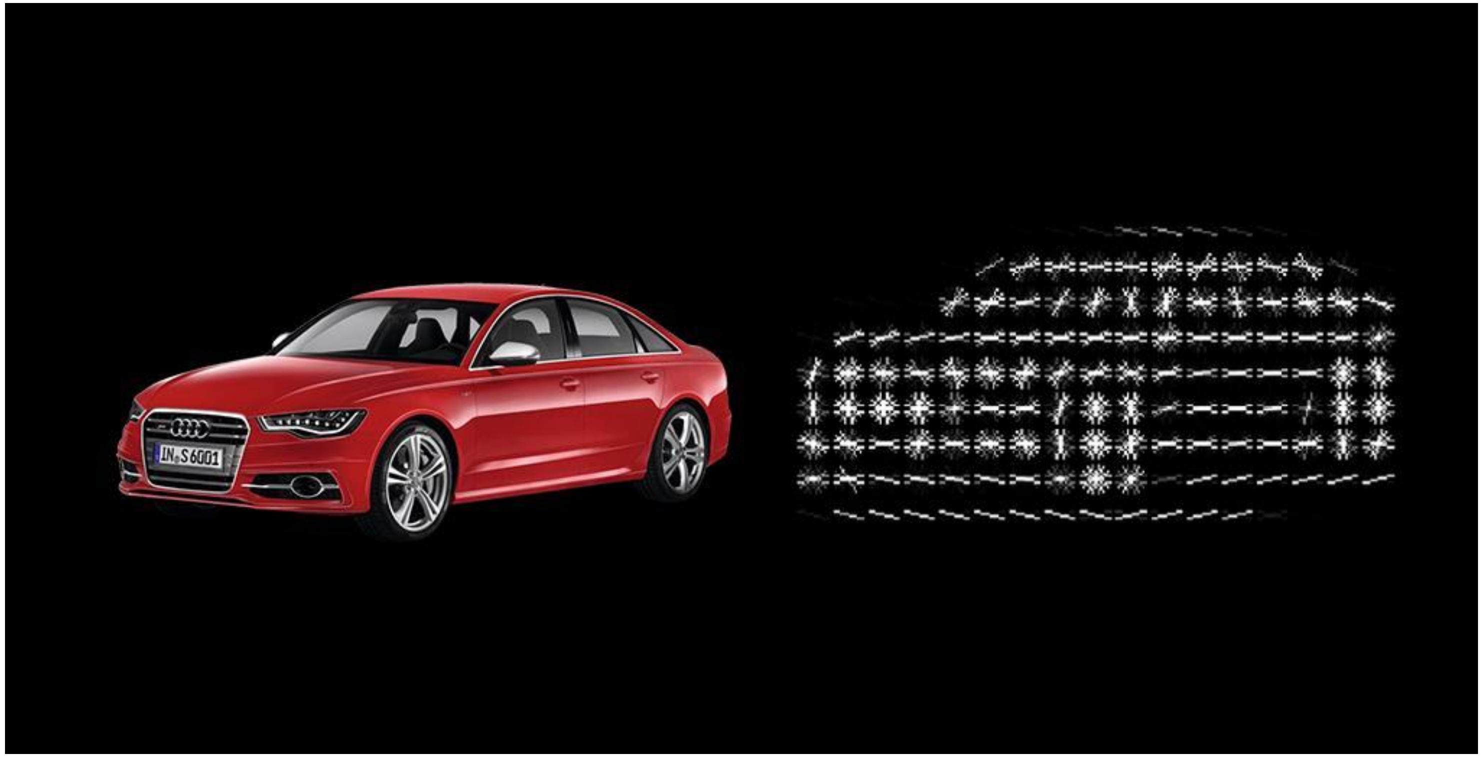
```
model = Sequential([
    Input(shape=(29, 68, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])
```

*internal structure inspired TheDeepHub

Keras Sequential Model

```
153/153 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.9033 - loss: 0.2569 - val_accuracy: 0.9655 - val_loss: 0.1186
Epoch 2/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9958 - loss: 0.0185 - val_accuracy: 0.9672 - val_loss: 0.1404
Epoch 3/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9965 - loss: 0.0086 - val_accuracy: 0.9770 - val_loss: 0.0563
Epoch 4/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 1.0000 - loss: 7.9550e-04 - val_accuracy: 0.9844 - val_loss: 0.0369
Epoch 5/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 1.0000 - loss: 2.8986e-04 - val_accuracy: 0.9819 - val_loss: 0.0455
Epoch 6/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 1.0000 - loss: 1.1871e-04 - val_accuracy: 0.9836 - val_loss: 0.0513
Epoch 7/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 1.0000 - loss: 2.0379e-04 - val_accuracy: 0.9836 - val_loss: 0.0387
Epoch 8/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - accuracy: 1.0000 - loss: 4.6520e-04 - val_accuracy: 0.9819 - val_loss: 0.0515
Epoch 9/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 1.0000 - loss: 7.4225e-05 - val_accuracy: 0.9836 - val_loss: 0.0461
Epoch 10/10
153/153 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 1.0000 - loss: 5.2256e-05 - val_accuracy: 0.9811 - val_loss: 0.0542
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9784 - loss: 0.0622
Validation Loss: 0.05424930527806282
Validation Accuracy: 0.9811165928840637
```

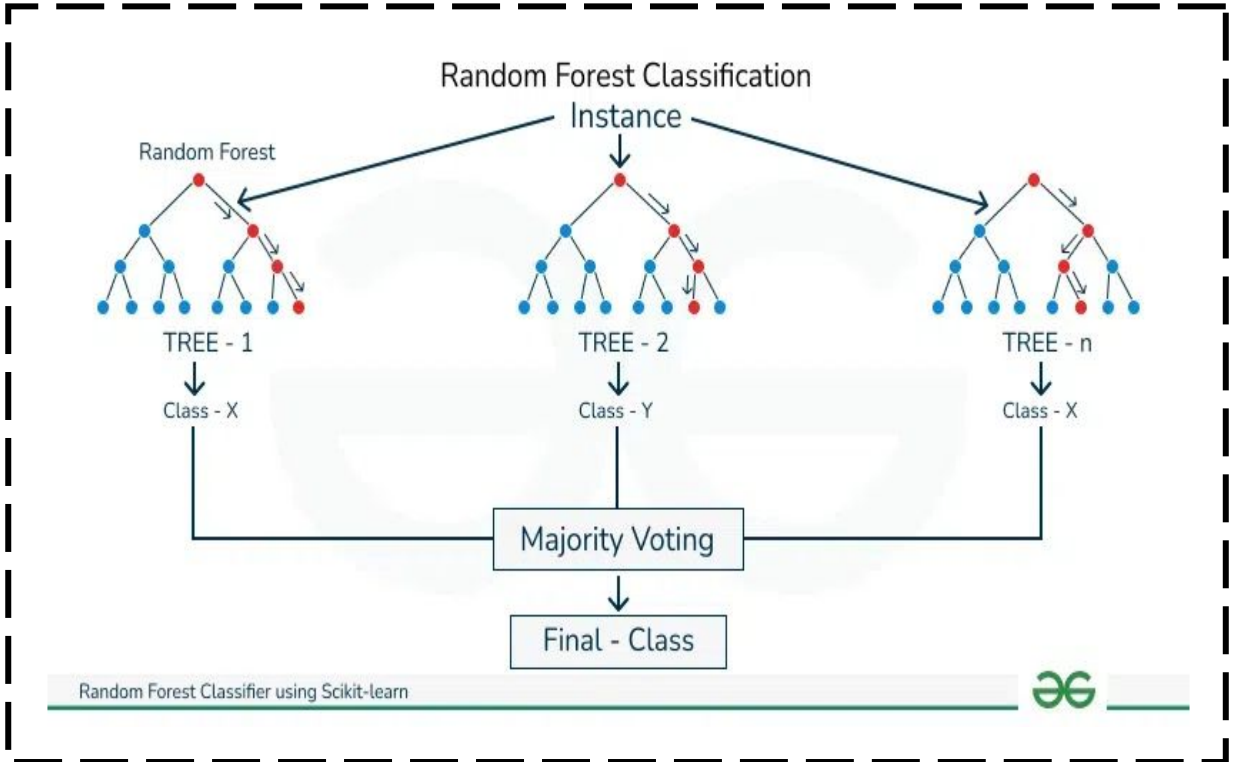
What about HOG Features?



Random Forest

- Combines the output of multiple decision trees to reach a single result

```
Random Forest Accuracy: 0.9961685823754789
Random Forest Loss: 0.09935789583867653
Random Forest Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       954
           2       1.00      1.00      1.00       873

    accuracy                           1.00      1827
   macro avg       1.00      1.00      1.00      1827
weighted avg       1.00      1.00      1.00      1827
```



SVC

- Classification tasks: finds the optimal hyperplane that separates different classes in the feature space

- Maximizing the margin between the classes to improve classification accuracy.

```
SVC Accuracy: 1.0
SVC Loss: 0.0005111067423619549
SVC Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       954
           2       1.00      1.00      1.00       873

    accuracy                           1.00      1827
   macro avg       1.00      1.00      1.00      1827
weighted avg       1.00      1.00      1.00      1827
```

4

Implementation

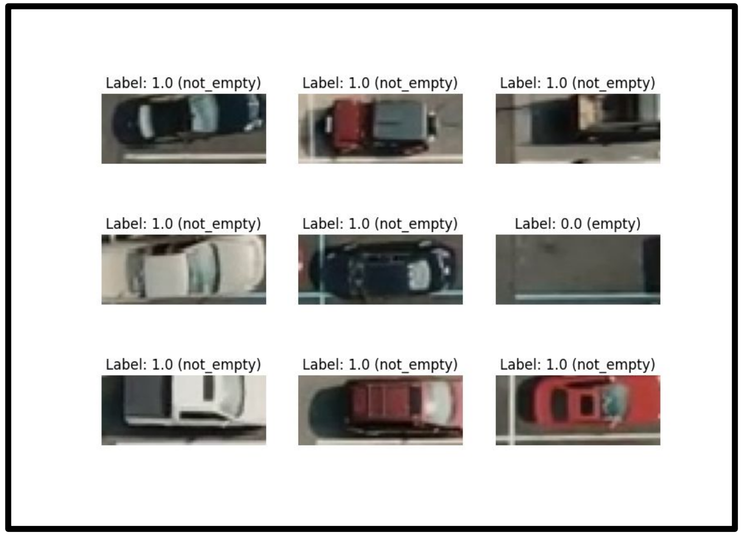
Creating Mask

```python
def draw_rectangle(event, x, y, flags, param):
    global ix, iy, drawing, img, mask
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            mask_copy = mask.copy()
            cv2.rectangle(mask_copy, (ix, iy), (x, y), (255, 255, 255), -1)
            display_img = cv2.addWeighted(img, 0.8, mask_copy, 0.2, 0)
            cv2.imshow("image", display_img)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        cv2.rectangle(mask, (ix, iy), (x, y), (255, 255, 255), -1)
        display_img = cv2.addWeighted(img, 0.8, mask, 0.2, 0)
        cv2.imshow("image", display_img)
```

```python
video_path = r"video.mp4"
cap = cv2.VideoCapture(video_path)

if not cap.isOpened():
    print("Error opening video file")
else:
    ret, frame = cap.read()  # Read the first frame
    if not ret:
        print("Failed to read the video")
    else:
        # Resize frame for convenience
        img = cv2.resize(frame, (800, 600))  # Resize for easier handling
        mask = np.zeros_like(img, dtype=np.uint8)

        cv2.namedWindow("image")
        cv2.setMouseCallback("image", draw_rectangle)

        display_img = cv2.addWeighted(img, 0.8, mask, 0.2, 0)
        while True:
            cv2.imshow("image", display_img)
            k = cv2.waitKey(1) & 0xFF
            if k == 27:  # ESC key to exit
                break

        cv2.imwrite("mask1.png", mask)
        cv2.destroyAllWindows()

cap.release()
```

Applying Mask to Video

Saving Keras Sequential Model



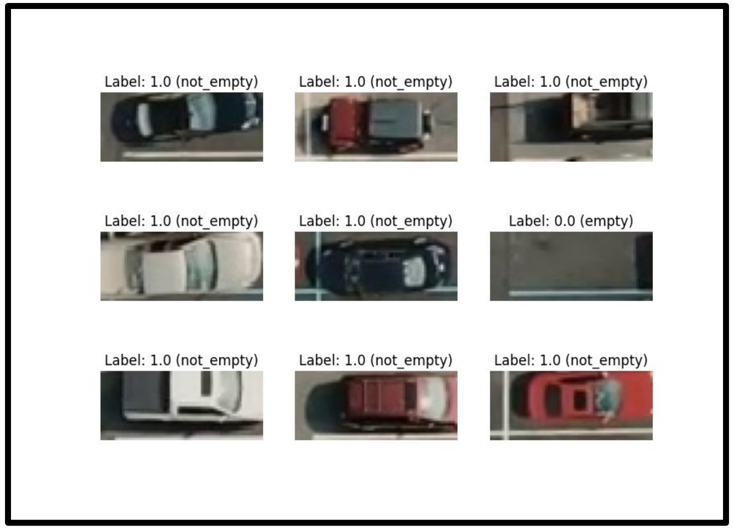

```
PS C:\Users\djibr\Desktop\EMAI\Semester 2 - Sapienza\Computer Vision\Computer Vision> python save_keras_sequential_model.py
Found 4872 images belonging to 2 classes.
Found 1218 images belonging to 2 classes.
Training labels: [1. 1. 1. 1. 1. 0. 1. 1. 1.]
2024-06-20 09:15:57.653512: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions
 in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler
flags.
Epoch 1/10
153/153 [==============================] - 15s 75ms/step - loss: 0.1259 - accuracy: 0.9557 - val_loss: 0.0905 - val_accuracy: 0.9663
Epoch 2/10
153/153 [==============================] - 12s 81ms/step - loss: 0.0152 - accuracy: 0.9955 - val_loss: 0.0848 - val_accuracy: 0.9754
Epoch 3/10
153/153 [==============================] - 13s 83ms/step - loss: 0.0108 - accuracy: 0.9967 - val_loss: 0.0443 - val_accuracy: 0.9819
Epoch 4/10
153/153 [==============================] - 13s 83ms/step - loss: 0.0085 - accuracy: 0.9973 - val_loss: 0.0376 - val_accuracy: 0.9860
Epoch 5/10
153/153 [==============================] - 10s 64ms/step - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.0456 - val_accuracy: 0.9869
Epoch 6/10
153/153 [==============================] - 9s 60ms/step - loss: 0.0041 - accuracy: 0.9984 - val_loss: 0.0117 - val_accuracy: 0.9967
Epoch 7/10
153/153 [==============================] - 9s 61ms/step - loss: 0.0038 - accuracy: 0.9990 - val_loss: 0.0467 - val_accuracy: 0.9860
Epoch 8/10
153/153 [==============================] - 9s 59ms/step - loss: 3.9094e-04 - accuracy: 1.0000 - val_loss: 0.0114 - val_accuracy: 0.9959
Epoch 9/10
153/153 [==============================] - 9s 61ms/step - loss: 1.7091e-04 - accuracy: 1.0000 - val_loss: 0.0175 - val_accuracy: 0.9918
Epoch 10/10
153/153 [==============================] - 9s 57ms/step - loss: 0.0015 - accuracy: 0.9994 - val_loss: 0.0074 - val_accuracy: 0.9959
39/39 [==============================] - 2s 47ms/step - loss: 0.0074 - accuracy: 0.9959
Validation Loss: 0.007370667532086372
Validation Accuracy: 0.9958949089050293
```

Bounding Box

```python
# Class to represent a bounding box, with the dimensions and border colour initialised
class BoundingBox:
    def __init__(self, x, y, w, h, class_id):
        self.x = x
        self.y = y
        self.w = w
        self.h = h
        self.class_id = class_id

    def draw(self, frame):
        color = (0, 255, 0) if self.class_id == 0 else (0, 0, 255)
        cv2.rectangle(
            frame, (self.x, self.y), (self.x + self.w, self.y + self.h), color, 2
        )
```

```python
video_path = r"video.mp4"
cap = cv2.VideoCapture(video_path)

if not cap.isOpened():
    print("Error opening video file")
else:
    ret, frame = cap.read()  # Read the first frame
    if not ret:
        print("Failed to read the video")
    else:
        # Resize frame for convenience
        img = cv2.resize(frame, (800, 600))  # Resize for easier handling
        mask = np.zeros_like(img, dtype=np.uint8)

        cv2.namedWindow("image")
        cv2.setMouseCallback("image", draw_rectangle)

        display_img = cv2.addWeighted(img, 0.8, mask, 0.2, 0)
        while True:
            cv2.imshow("image", display_img)
            k = cv2.waitKey(1) & 0xFF
            if k == 27:  # ESC key to exit
                break

        cv2.imwrite("mask1.png", mask)
        cv2.destroyAllWindows()

cap.release()
```

Saving MobileNetV2 Model




```
PS C:\Users\djibr\Desktop\EMAI\Semester 2 - Sapienza\Computer Vision\Computer Vision> python save_mobilenet2_model.py
Found 4872 images belonging to 2 classes.
Found 1218 images belonging to 2 classes.
Training labels: [[0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]]
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
2024-06-20 09:20:47.969910: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions
 in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/10
153/153 [==============================] - 16s 75ms/step - loss: 0.1152 - accuracy: 0.9641 - val_loss: 0.1381 - val_accuracy: 0.9532
Epoch 2/10
153/153 [==============================] - 11s 72ms/step - loss: 0.0252 - accuracy: 0.9959 - val_loss: 0.1126 - val_accuracy: 0.9622
Epoch 3/10
153/153 [==============================] - 10s 61ms/step - loss: 0.0139 - accuracy: 0.9977 - val_loss: 0.0885 - val_accuracy: 0.9778
Epoch 4/10
153/153 [==============================] - 14s 92ms/step - loss: 0.0093 - accuracy: 0.9990 - val_loss: 0.0956 - val_accuracy: 0.9639
Epoch 5/10
153/153 [==============================] - 12s 79ms/step - loss: 0.0069 - accuracy: 0.9992 - val_loss: 0.1016 - val_accuracy: 0.9631
Epoch 6/10
153/153 [==============================] - 10s 62ms/step - loss: 0.0045 - accuracy: 1.0000 - val_loss: 0.0817 - val_accuracy: 0.9787
Epoch 7/10
153/153 [==============================] - 12s 79ms/step - loss: 0.0038 - accuracy: 1.0000 - val_loss: 0.0834 - val_accuracy: 0.9745
Epoch 8/10
153/153 [==============================] - 12s 75ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.0867 - val_accuracy: 0.9713
Epoch 9/10
153/153 [==============================] - 11s 74ms/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 0.0899 - val_accuracy: 0.9663
Epoch 10/10
153/153 [==============================] - 11s 74ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0809 - val_accuracy: 0.9803
39/39 [==============================] - 2s 59ms/step - loss: 0.0809 - accuracy: 0.9803
Validation Loss: 0.08088847249746323
Validation Accuracy: 0.9802955389022827
```

Resized Training Images

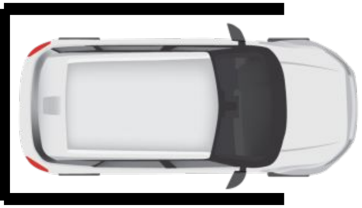
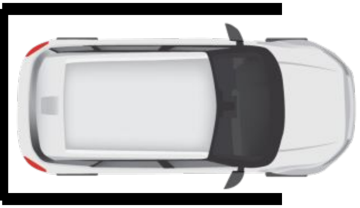
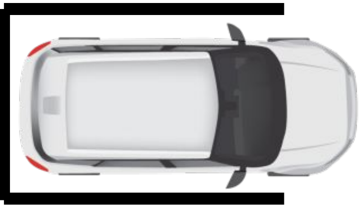
Keras Sequential

```
# Function to resise the desired area for the bounding box to the target size that is specific to the training model used
def preprocess_for_prediction(desired_area, target_size=(68, 29)):
    desired_area_resized = cv2.resize(desired_area, target_size)
    desired_area_normalized = desired_area_resized / 255.0
    desired_area_expanded = np.expand_dims(desired_area_normalized, axis=0)
    return desired_area_expanded
```

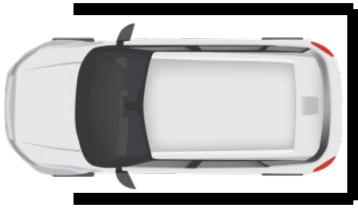
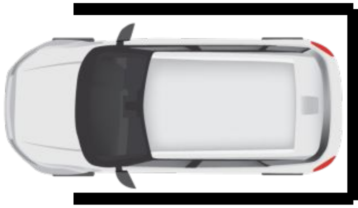
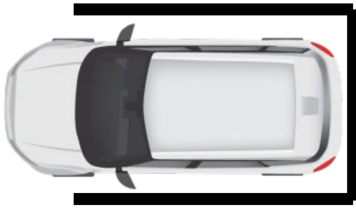
MobileNetV2

```
# Function to resise the desired area for the bounding box to the target size that is specific to the training model used
def preprocess_for_prediction(desired_area, target_size=(75, 32)):
    desired_area_resized = cv2.resize(desired_area, target_size)
    desired_area_normalized = desired_area_resized / 255.0
    desired_area_expanded = np.expand_dims(desired_area_normalized, axis=0)
    return desired_area_expanded
```

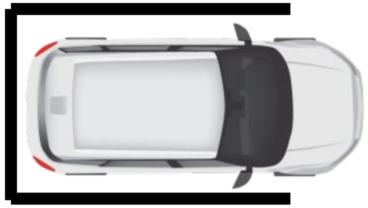
Predicting Bounding Box

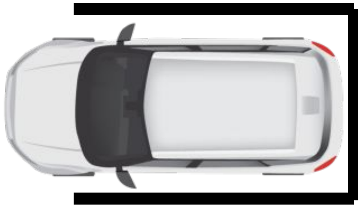




```
# Function to draw the bounding boxes on an input frame and predict the occupancy of each box
def draw_bounding_boxes_and_predict(frame, mask, model, bounding_boxes):
    # Turn frame into black and white and mask the different regions/features of the frame
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    segmented = cv2.bitwise_and(gray, gray, mask=mask)
    contours, _ = cv2.findContours(
        segmented, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )

    # Finding contours of parking space and add the bounding box to it
    desired_areas = []
    new_bounding_boxes = []
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)
        desired_area = frame[y : y + h, x : x + w]
        desired_area_preprocessed = preprocess_for_prediction(desired_area)
        desired_areas.append(desired_area_preprocessed)
        new_bounding_boxes.append(BoundingBox(x, y, w, h, 0))  # Temporary class_id=0

    # Predict occupancy on parking space using trained model
    if desired_areas:
        desired_areas_batch = np.vstack(desired_areas)
        predictions = model.predict(desired_areas_batch)
        predicted_classes = np.argmax(predictions, axis=1)

        for i, bounding_box in enumerate(new_bounding_boxes):
            bounding_box.class_id = predicted_classes[i]

    # Update and draw bounding boxes
    bounding_boxes.clear()
    bounding_boxes.extend(new_bounding_boxes)
    for bounding_box in bounding_boxes:
        bounding_box.draw(frame)

    return frame
```

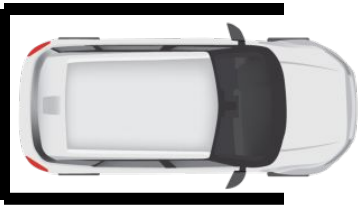




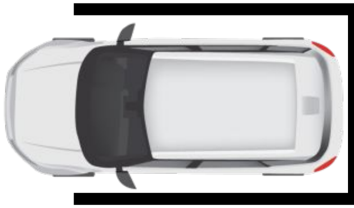
Non-Occupied Parking Spaces





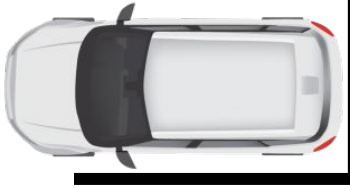
```python
# Function to display number of non-occupied parking spaces
def draw_empty_spots_counter(frame, bounding_boxes):
    total_spots = len(bounding_boxes)
    empty_spots = sum(bbox.class_id == 0 for bbox in bounding_boxes)
    text = f"Empty Spots: {empty_spots}/{total_spots}"

    # Counter Dimentions
    box_x, box_y = 10, 10
    box_w, box_h = 200, 50

    # Draw counter box and apply text
    cv2.rectangle(
        frame, (box_x, box_y), (box_x + box_w, box_y + box_h), (255, 255, 255), -1
    )
    cv2.putText(
        frame,
        text,
        (box_x + 10, box_y + 30),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        (0, 0, 0),
        2,
    )

    return frame
```





SVC/Random Forest

Load Images/Labels



```python
# Function to load images and labels from the dataset folder
def load_images(folder, target_size=(68, 29)):
    images = []
    labels = []
    for label, subfolder in enumerate(os.listdir(folder)):
        subfolder_path = os.path.join(folder, subfolder)
        if os.path.isdir(subfolder_path):
            for filename in os.listdir(subfolder_path):
                img_path = os.path.join(subfolder_path, filename)
                if filename.endswith((".png", ".jpg", ".jpeg", ".bmp", ".gif")):
                    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

                    # Resize image to correct target size
                    if img is not None:
                        img = cv2.resize(img, target_size)
                        images.append(img)
                        labels.append(label)
    return images, labels
```

```python
# Load dataset
dataset_folder = "dataset"
images, labels = load_images(dataset_folder)
```

Extract Hog Feature

```python
# Function to extract HOG features from the images obtained in the dataset
def extract_hog_features(images):
    hog_features = []
    for image in images:
        feature = hog(
            image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=False
        )
        hog_features.append(feature)
    return np.array(hog_features)
```

```python
# Fucntion call to extract HOG features
hog_features = extract_hog_features(images)
```

Train and Predict

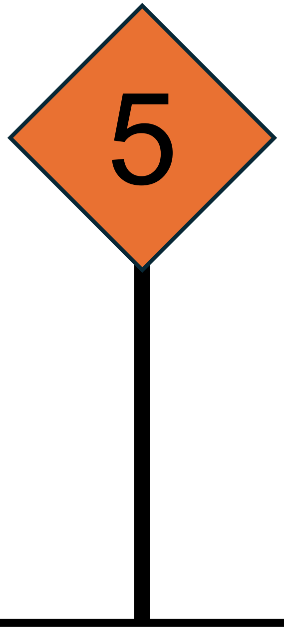
```
# Splitting training and testing data
X_train, X_test, y_train, y_test = train_test_split(
    hog_features, labels, test_size=0.3, random_state=42
)
```

SVC

```
################################
# SVC with GridSearchCV        #
################################

# Train the model
svc_params = {"kernel": ("linear", "rbf"), "C": [1, 10]}
svc = SVC(probability=True)
svc_clf = GridSearchCV(svc, svc_params)
svc_clf.fit(X_train, y_train)


# Predict and display the evaluation of the model
svc_best = svc_clf.best_estimator_
svc_predictions = svc_best.predict(X_test)
svc_probabilities = svc_best.predict_proba(X_test)
svc_accuracy = accuracy_score(y_test, svc_predictions)
svc_loss = log_loss(y_test, svc_probabilities)
print("SVC Accuracy:", svc_accuracy)
print("SVC Loss:", svc_loss)
print("SVC Classification Report:\n", classification_report(y_test, svc_predictions))
```

Random Forest

```
################################
# Random Forest with GridSearchCV #
################################

# Train the model
rf_params = {"n_estimators": [50, 100, 200], "max_depth": [None, 10, 20, 30]}
rf = RandomForestClassifier()
rf_clf = GridSearchCV(rf, rf_params)
rf_clf.fit(X_train, y_train)

# Predict and display the evaluation of the model
rf_best = rf_clf.best_estimator_
rf_predictions = rf_best.predict(X_test)
rf_probabilities = rf_best.predict_proba(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_loss = log_loss(y_test, rf_probabilities)
print("Random Forest Accuracy:", rf_accuracy)
print("Random Forest Loss:", rf_loss)
print(
    "Random Forest Classification Report:\n",
    classification_report(y_test, rf_predictions),
)
```



Evaluation

1

Keras Sequential

```
Validation Loss: 0.007370667532086372
Validation Accuracy: 0.9958949089050293
```

2

MobileNetV2

```
Validation Loss: 0.0808847249746323
Validation Accuracy: 0.9802955389022827
```

3

SVC

```
SVC Accuracy: 1.0
SVC Loss: 0.0005111067423619549
SVC Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       954
           2       1.00      1.00      1.00       873

    accuracy                           1.00      1827
   macro avg       1.00      1.00      1.00      1827
weighted avg       1.00      1.00      1.00      1827
```

4

Random Forest

```
Random Forest Accuracy: 0.9961685823754789
Random Forest Loss: 0.09935789583867653
Random Forest Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       954
           2       1.00      1.00      1.00       873

    accuracy                           1.00      1827
   macro avg       1.00      1.00      1.00      1827
weighted avg       1.00      1.00      1.00      1827
```

Demonstration of Project

Live Demonstration done in class

See GitHub Repository to run the project

6

Conclusion

What we’ve learned

1

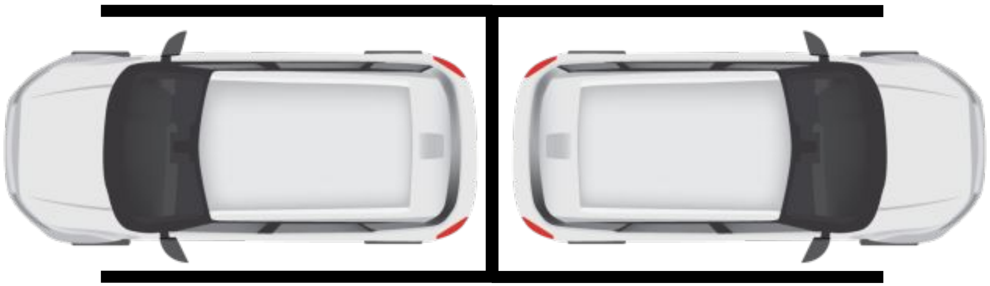
Basic Classification Algorithms **outperformed** Convoluted models based on project **objectives**

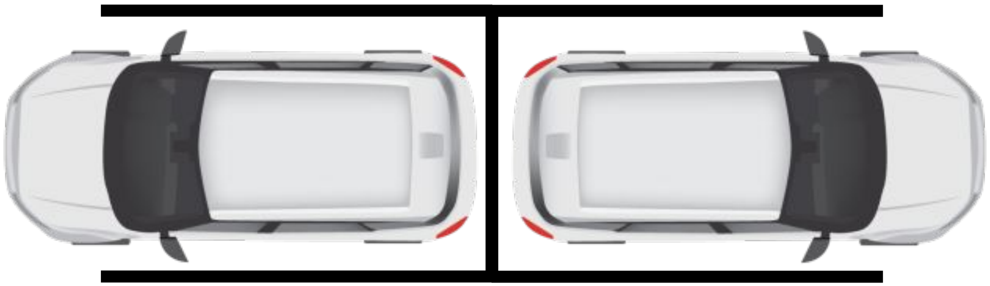
2

Quality and **pre-processing** of the dataset/labels is important

3

Future improvements **(Use of IoT devices, real-time camera feed)**





Thank You

Any Questions