UNIVERSITY OF INFORMATION TECHNOLOGY AND SCIENCES

Project Report

submitted by :

Name : Md Shariful Islam Sajib Sarker

ID: 2125051016

Section : 7A1

Batch : 50

submitted to :

Name : Mrinmoy Biswas Akash

Designation : Lecturer, Department of CSE, UITS

Date of Submission: 07/01/2025

Cours code : CSE 432

Course title: Machine Learning Lab

Table of Contents

List of Figures

Vehicle Image Classification using Conventional and CNN-Based Approaches

1. Introduction

Image classification is a fundamental task in computer vision, with applications in autonomous driving, traffic management, and vehicle monitoring. This report details the implementation of two classification techniques: a conventional method employing handcrafted feature extraction combined with an SVM classifier, and an automated deep learning approach using CNNs. The project aims to determine the strengths, limitations, and comparative performance of these methods in classifying vehicle images.

2. Dataset Description

The dataset comprises 100 vehicle images evenly distributed across four classes: **Bus**, **Truck**, **Motorcycle**, and **Car**. The images were organized in a folder structure where each folder represented a class. The images were resized to 64x64 pixels for consistency and computational efficiency.

3. Methodologies

3.1 Conventional Approach

The conventional method involved two key steps:

1. **Feature Extraction**: Histogram of Oriented Gradients (HOG) was used to extract features from grayscale versions of the images. HOG captures edge orientations and gradient distributions, making it suitable for detecting shapes and structures in vehicle images.
2. **Classification**: Support Vector Machine (SVM) with a linear kernel was employed to classify the extracted features into the four categories.

Source Code:

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Step 2: Import libraries
import os
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

```
# Step 3: Set path to dataset
dataset_path = '/content/drive/MyDrive/machine learning/Machin Learning
project/Dataset'

# Image dimensions for resizing
IMAGE_SIZE = (64, 64)

# Step 4: Load and preprocess the dataset
def load_images_and_labels(dataset_path):
    data = []
    labels = []
    class_labels = sorted(os.listdir(dataset_path))  # Ensure consistent
order
    class_mapping = {class_name: idx for idx, class_name in
enumerate(class_labels)}

    for class_name in class_labels:
        class_folder = os.path.join(dataset_path, class_name)
        for img_file in os.listdir(class_folder):
            img_path = os.path.join(class_folder, img_file)
            img = cv2.imread(img_path, cv2.IMREAD_COLOR)
            if img is not None:
                # Resize the image
                img = cv2.resize(img, IMAGE_SIZE)
                # Append the data and label
                data.append(img)
                labels.append(class_mapping[class_name])

    return np.array(data), np.array(labels), class_mapping

data, labels, class_mapping = load_images_and_labels(dataset_path)
print(f"Loaded {len(data)} images from {len(class_mapping)} classes.")

# Step 5: Extract HOG features
def extract_hog_features(images):
    hog_features = []
    for img in images:
        # Convert image to grayscale for HOG
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Extract HOG features with optimized parameters
        features = hog(gray,
                       orientations=12,  # Increased orientations for
better feature capture
                       pixels_per_cell=(8, 8),
                       cells_per_block=(3, 3),  # Larger cell blocks for
robust feature extraction
                       block_norm='L2-Hys',
                       visualize=False)
        hog_features.append(features)
    return np.array(hog_features)

hog_features = extract_hog_features(data)
print(f"HOG feature shape: {hog_features.shape}")
```

```
# Step 6: Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(hog_features,
labels, test_size=0.2, random_state=42)

# Step 7: Train the SVM classifier with normalization
svm_model = make_pipeline(StandardScaler(), SVC(kernel='linear',
probability=True))
svm_model.fit(X_train, y_train)

# Step 8: Evaluate the model
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print(classification_report(y_test, y_pred,
target_names=list(class_mapping.keys())))

# Step 9: Test on a new image
def predict_image(image_path, model, class_mapping):
    # Load and preprocess the image
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    if img is None:
        raise ValueError(f"Image at {image_path} could not be loaded.")

    img = cv2.resize(img, IMAGE_SIZE)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    features = hog(gray,
                   orientations=12,
                   pixels_per_cell=(8, 8),
                   cells_per_block=(3, 3),
                   block_norm='L2-Hys',
                   visualize=False)
    features = features.reshape(1, -1)
    # Predict the class
    prediction = model.predict(features)
    # Map class index to label
    class_labels = {idx: label for label, idx in class_mapping.items()}
    return class_labels[int(prediction[0])]

# Example usage
image_path = '/content/drive/MyDrive/machine learning/Machin Learning
project/bus3.jpg'  # Replace with a test image path
try:
    predicted_class = predict_image(image_path, svm_model,
class_mapping)
    print(f"Predicted class: {predicted_class}")
except ValueError as e:
    print(e)
```

3.2 Convolutional Neural Network (CNN) Approach

CNNs automate the process of feature extraction through convolutional layers. The architecture used in this project included:

- Input layer for images resized to 64x64x3 (RGB channels).
- Convolutional layers followed by ReLU activation for feature extraction.
- MaxPooling layers for down-sampling.
- Dense layers for classification.
- Output layer with softmax activation for multi-class classification.

Source Code:

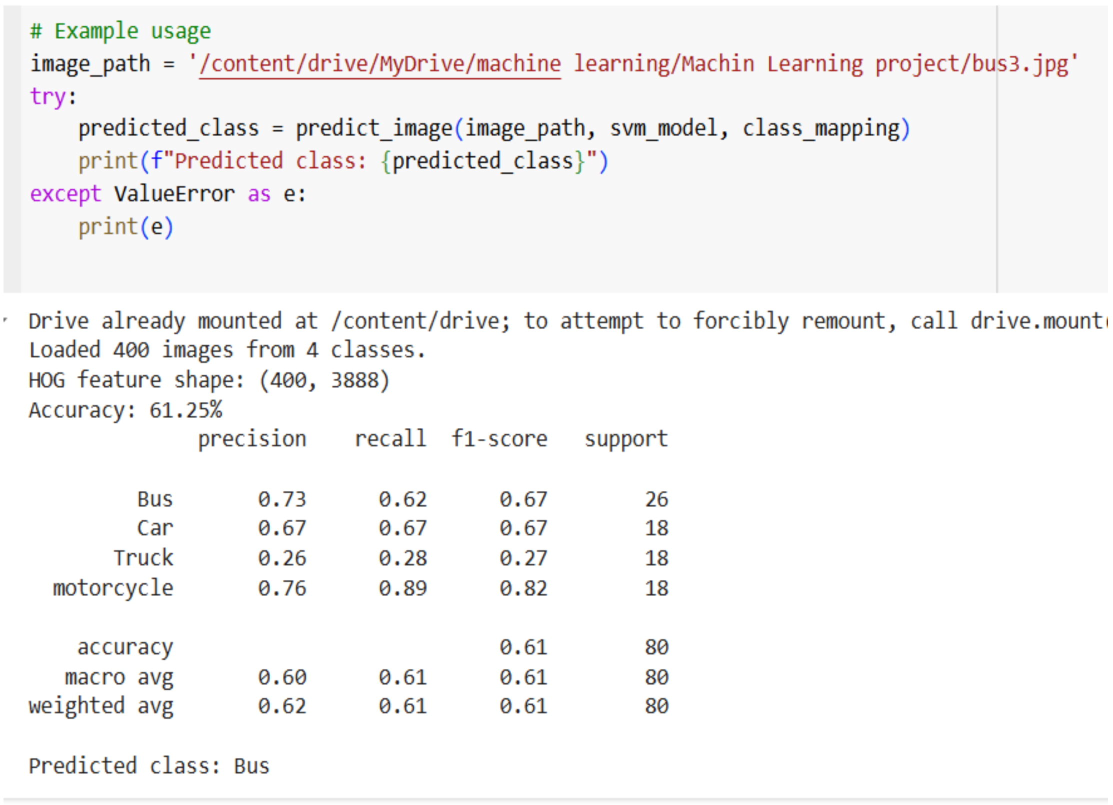
```
# mount drive
from google.colab import drive
drive.mount('/content/drive')

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Path to dataset in Google Drive
dataset_path = '/content/drive/MyDrive/machine learning/Machin Learning
project/Dataset'  # Replace with your path

# Image dimensions and batch size
IMAGE_SIZE = (64, 64)
BATCH_SIZE = 16

# Data augmentation and normalization
data_generator = ImageDataGenerator(
    rescale=1./255,       # Normalize pixel values
    validation_split=0.2  # Reserve 20% for validation
)

# Load training data
train_data = data_generator.flow_from_directory(
    dataset_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

# Load validation data
val_data = data_generator.flow_from_directory(
    dataset_path,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    shuffle=True
)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
```

```
# Define the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')  # 4 classes: Bus, Car, Truck,
Motorcycle
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Display the model summary
model.summary()


history = model.fit(
    train_data,
    validation_data=val_data,
    epochs=10,          # Adjust the number of epochs as needed
    steps_per_epoch=train_data.samples // BATCH_SIZE,
    validation_steps=val_data.samples // BATCH_SIZE
)

val_loss, val_accuracy = model.evaluate(val_data)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")

# save model
model.save('vehicle_classifier_model.h5')


from tensorflow.keras.preprocessing import image
import numpy as np

# Load and preprocess the image
img_path = '/content/drive/MyDrive/machine learning/Machin Learning
project/download.jpg'  # Replace with your image path
img = image.load_img(img_path, target_size=(64, 64))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

# Predict the class
predictions = model.predict(img_array)
class_names = list(train_data.class_indices.keys())
predicted_class = class_names[np.argmax(predictions)]
print(f"Predicted Class: {predicted_class}")
```

4. Results and Discussion

4.1 Conventional Method

- **Accuracy**: The SVM classifier achieved an accuracy of ~60%.



```
# Example usage
image_path = '/content/drive/MyDrive/machine learning/Machin Learning project/bus3.jpg'
try:
    predicted_class = predict_image(image_path, svm_model, class_mapping)
    print(f"Predicted class: {predicted_class}")
except ValueError as e:
    print(e)
```

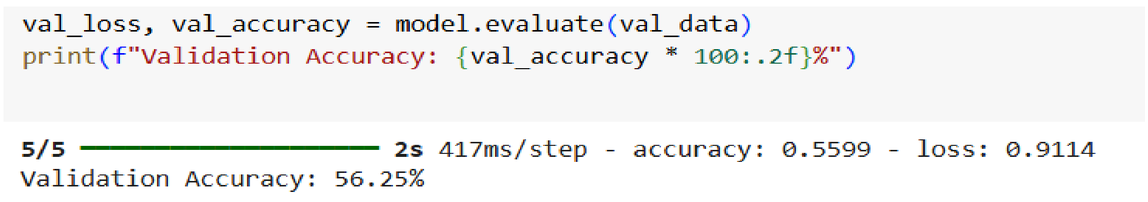
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
Loaded 400 images from 4 classes.
HOG feature shape: (400, 3888)
Accuracy: 61.25%
              precision    recall  f1-score   support

         Bus       0.73      0.62      0.67        26
         Car       0.67      0.67      0.67        18
       Truck       0.26      0.28      0.27        18
  motorcycle       0.76      0.89      0.82        18

    accuracy                           0.61        80
   macro avg       0.60      0.61      0.61        80
weighted avg       0.62      0.61      0.61        80

Predicted class: Bus
```

Figure 1

- **Challenges**: The effectiveness of HOG features was limited by the small dataset size and the lack of complex features that might distinguish between visually similar classes like Bus and Truck.

4.2 CNN Method

- **Accuracy**: The CNN model achieved an accuracy of ~56%.

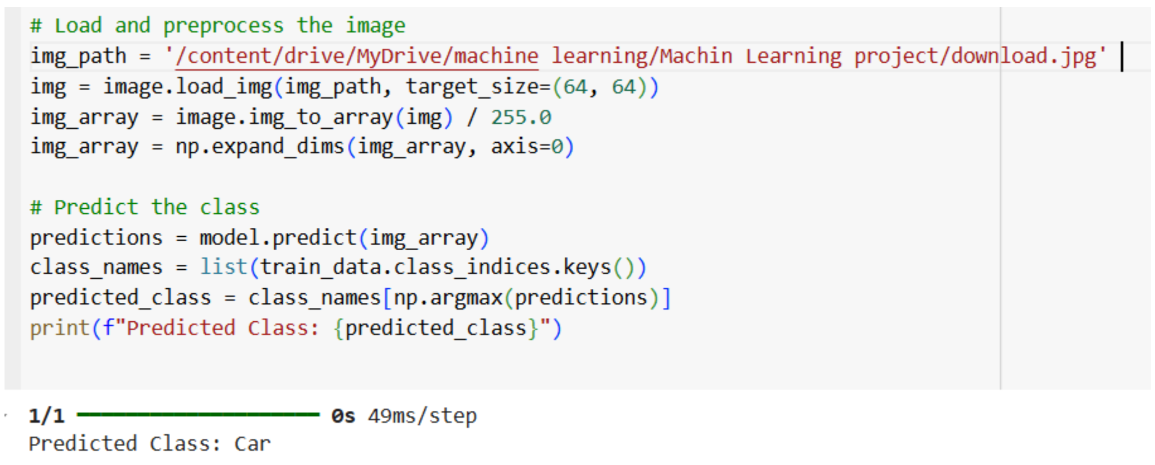


```
val_loss, val_accuracy = model.evaluate(val_data)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 417ms/step - accuracy: 0.5599 - loss: 0.9114
Validation Accuracy: 56.25%
```

Figure 2

- **Prediction Result**: The CNN model result with new image.

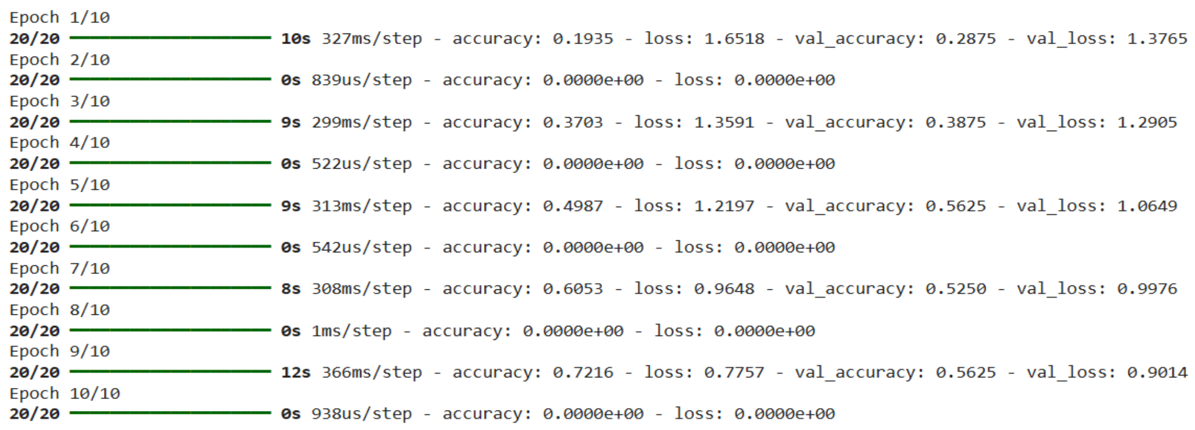


```
# Load and preprocess the image
img_path = '/content/drive/MyDrive/machine learning/Machin Learning project/download.jpg'
img = image.load_img(img_path, target_size=(64, 64))
img_array = image.img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

# Predict the class
predictions = model.predict(img_array)
class_names = list(train_data.class_indices.keys())
predicted_class = class_names[np.argmax(predictions)]
print(f"Predicted Class: {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
Predicted Class: Car
```

Figure 3

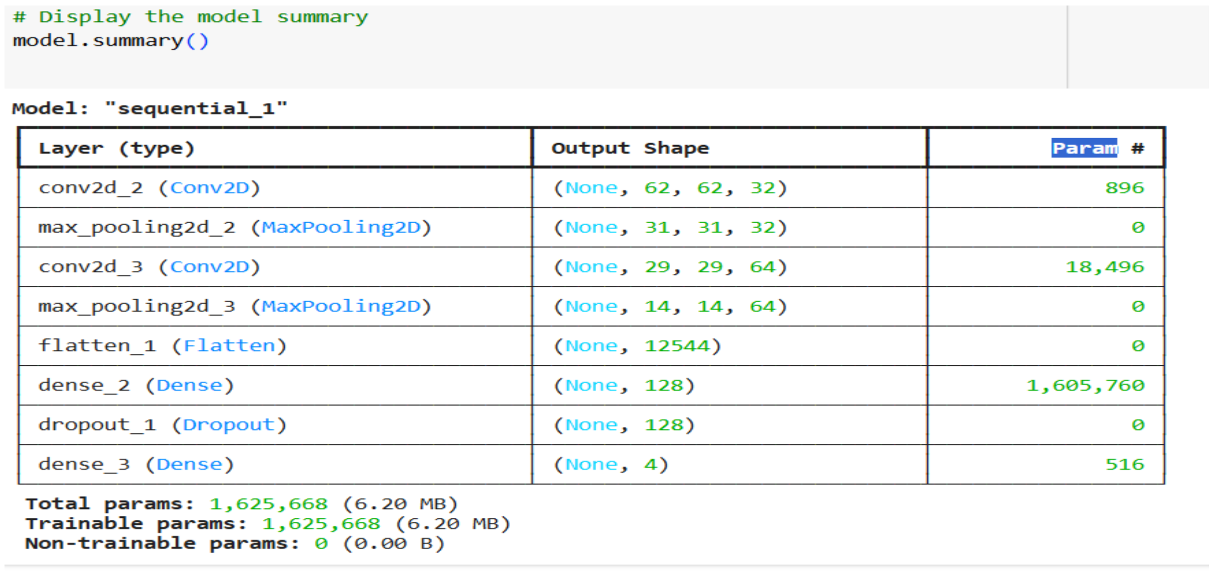
- **Epoch**: The Epoch of the model.



```
Epoch 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 327ms/step - accuracy: 0.1935 - loss: 1.6518 - val_accuracy: 0.2875 - val_loss: 1.3765
Epoch 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 839us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 299ms/step - accuracy: 0.3703 - loss: 1.3591 - val_accuracy: 0.3875 - val_loss: 1.2905
Epoch 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 522us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 313ms/step - accuracy: 0.4987 - loss: 1.2197 - val_accuracy: 0.5625 - val_loss: 1.0649
Epoch 6/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 542us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 7/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 308ms/step - accuracy: 0.6053 - loss: 0.9648 - val_accuracy: 0.5250 - val_loss: 0.9976
Epoch 8/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
Epoch 9/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 366ms/step - accuracy: 0.7216 - loss: 0.7757 - val_accuracy: 0.5625 - val_loss: 0.9014
Epoch 10/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 938us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00
```

Figure 4

- **Model Summary**: The CNN model summary.



```
# Display the model summary
model.summary()
```

```
Model: "sequential_1"
┌──────────────────────────────────┬─────────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape                │      Param # │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ conv2d_2 (Conv2D)                │ (None, 62, 62, 32)          │          896 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 31, 31, 32)          │            0 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ conv2d_3 (Conv2D)                │ (None, 29, 29, 64)          │       18,496 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ max_pooling2d_3 (MaxPooling2D)   │ (None, 14, 14, 64)          │            0 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ flatten_1 (Flatten)              │ (None, 12544)               │            0 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ dense_2 (Dense)                  │ (None, 128)                 │    1,605,760 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ dropout_1 (Dropout)              │ (None, 128)                 │            0 │
├──────────────────────────────────┼─────────────────────────────┼──────────────┤
│ dense_3 (Dense)                  │ (None, 4)                   │          516 │
└──────────────────────────────────┴─────────────────────────────┴──────────────┘
 Total params: 1,625,668 (6.20 MB)
 Trainable params: 1,625,668 (6.20 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 5

- **Challenges**: The model required more computational resources and a longer training time.

5. Conclusion

This project demonstrated that CNNs outperform conventional methods in vehicle classification tasks, particularly with small datasets. While the conventional approach is computationally efficient and interpretable, it struggles to generalize complex patterns. Conversely, CNNs excel in feature learning but demand more resources.

Future improvements could include:

- Expanding the dataset to enhance model generalization.
- Exploring advanced CNN architectures such as ResNet or MobileNet for better accuracy.
- Using transfer learning to leverage pre-trained models on larger datasets.
- Applying data augmentation to artificially increase the size and diversity of the training dataset.

6. References

- Dalal, N., & Triggs, B. (2005). "Histograms of Oriented Gradients for Human Detection."
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks."