In [2]:
```python
import tensorflow as tf

# Download pre-trained InceptionV3 model
inceptionv3 = tf.keras.applications.InceptionV3(weights='imagenet')
```

In [3]:
```python
# Load the pre-trained inceptionv3 model

import tensorflow as tf

# Load pre-trained InceptionV3 model without top classification layer
inceptionv3 = tf.keras.applications.InceptionV3(weights='imagenet', include_top=False)

# Remove the last layer from the model
inceptionv3 = tf.keras.Model(inputs=inceptionv3.input, outputs=inceptionv3.layers[-1].output)
```

In [4]:
```python
inceptionv3
```

Out[4]:
```
<keras.engine.functional.Functional at 0x1c28c713430>
```

In [5]:
```python
# Define the input size of the VGG16 model
input_size = (224, 224)

# Specify the path to the PASCAL VOC 2007 dataset
data = 'C:\\Users\\Admin\\Downloads\\VOCdevkit\\VOC2007'
```

In [ ]:
```python

```

In [6]:
```
import os
import numpy as np
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

In [7]:
```
# Specify the path to the training image folder
train_image_folder = os.path.join(data, 'JPEGImages')
train_image_folder
```

Out[7]:
```
'C:\\Users\\Admin\\Downloads\\VOCdevkit\\VOC2007\\JPEGImages'
```

In [8]:
```
# Get the list of image filenames in the training set
train_image_filenames = os.listdir(train_image_folder)
```

In [9]:

```python
# Load and preprocess the images in the PASCAL VOC 2007 training set
def preprocess_image(image_path):
    img = load_img(image_path, target_size=input_size)
    img = img_to_array(img)
    img = preprocess_input(img)
    return img

# Process each image in the training set and extract features
training_features = []
for filename in train_image_filenames:
    image_path = os.path.join(train_image_folder, filename)
    img = preprocess_image(image_path)
    features = inceptionv3.predict(tf.expand_dims(img, axis=0))
    training_features.append(features)

# Concatenate the extracted features into a single array
training_features = tf.concat(training_features, axis=0)
```

```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 82ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 82ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 62ms/step
```

In [10]:
```
# Save the features to a NumPy array
np.save('features.npy', training_features)
```

In [11]:
```
# Specify the path to the annotations folder
annotation_folder = os.path.join(data, 'Annotations')
```

In [12]:
```
# Get the list of XML annotation filenames
annotation_filenames = os.listdir(annotation_folder)
```

In [13]:
```
# Process each XML annotation file and extract the label names

import xml.etree.ElementTree as ET

label_names = []
for filename in annotation_filenames:
    annotation_path = os.path.join(annotation_folder, filename)
    tree = ET.parse(annotation_path)
    root = tree.getroot()
    for obj in root.findall('object'):
        class_name = obj.find('name').text
        label_names.append(class_name)

# Get unique label names
unique_label_names = list(set(label_names))
```

In [24]:
```
# Create a dictionary to map unique label names to label indices
class_to_label = {label_name: label_index for label_index, label_name in enumerate(unique_label_names)}

# Process each XML annotation file again and extract the labels
labels = []
labels_bin = []
for filename in annotation_filenames:
    labels=[]
    annotation_path = os.path.join(annotation_folder, filename)
    tree = ET.parse(annotation_path)
    root = tree.getroot()
    for obj in root.findall('object'):
        class_name = obj.find('name').text
        label = class_to_label[class_name]
        labels.append(label)
    #test.append(labels)
    labels_bin.append(labels)


print(labels_bin)
```

...

In [14]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
# Load the features and labels
features = np.load('features.npy')
labels = np.load('labels.npy')
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(features, labels, test_size=0.2, random_state=42)
```

In [ ]:

In [15]:
```
len(X_train), len(X_val), len(y_train), len(y_val)
```

Out[15]:
```
(3961, 991, 3961, 991)
```

In [16]:
```
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

Out[16]:
```
((3961, 5, 5, 2048), (991, 5, 5, 2048), (3961, 20), (991, 20))
```

In [23]:
```
# Reshape training data
n_train_samples = X_train.shape[0]
X_train_2d = X_train.reshape(n_train_samples, -1)  # Reshape to (n_train_samples, 5*5*2048)
y_train_2d = y_train.argmax(axis=1)  # Convert one-hot encoded labels to single dimension

# Reshape validation data
n_val_samples = X_val.shape[0]
X_val_2d = X_val.reshape(n_val_samples, -1)  # Reshape to (n_val_samples, 5*5*2048)
y_val_2d = y_val.argmax(axis=1)  # Convert one-hot encoded labels to single dimension
```

In [25]:
```
X_train_2d.shape, X_val_2d.shape, y_train_2d.shape, y_val_2d .shape
```

Out[25]:
```
((3961, 51200), (991, 51200), (3961,), (991,))
```

In [33]:
```
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [34]:

```python
# Get the number of classes from the training labels
num_classes = len(np.unique(y_train_2d))

# Create a list to store the SVM classifiers
svm_classifiers = []

# Train binary SVM classifiers for each class
for class_label in range(num_classes):
    # Create an SVM classifier object
    svm_classifier = svm.SVC(kernel='linear')

    # Prepare binary labels for the current class
    binary_train_labels = (y_train_2d == class_label)
    binary_val_labels = (y_val_2d == class_label)

    # Train the SVM classifier
    svm_classifier.fit(X_train_2d, binary_train_labels)

    # Predict on the validation set
    val_predictions = svm_classifier.predict(X_val_2d)

    # Calculate accuracy
    accuracy = accuracy_score(binary_val_labels, val_predictions)
    print(f"Accuracy for class {class_label}: {accuracy}")

    # Store the trained classifier
    svm_classifiers.append(svm_classifier)
```

```
Accuracy for class 0: 0.9525731584258325
Accuracy for class 1: 0.9919273461150353
Accuracy for class 2: 0.9576185671039354
Accuracy for class 3: 0.941473259334006
Accuracy for class 4: 0.9818365287588294
Accuracy for class 5: 0.9757820383451059
Accuracy for class 6: 0.9162462159434914
Accuracy for class 7: 0.9626639757820383
Accuracy for class 8: 0.9828456104944501
Accuracy for class 9: 0.9889001009081736
Accuracy for class 10: 0.992936427850656
Accuracy for class 11: 0.9727547931382442
Accuracy for class 12: 0.987891019172553
Accuracy for class 13: 0.9889001009081736
Accuracy for class 14: 0.9899091826437941
Accuracy for class 15: 0.887991927346115
Accuracy for class 16: 0.987891019172553
Accuracy for class 17: 0.9899091826437941
Accuracy for class 18: 0.9828456104944501
Accuracy for class 19: 0.9868819374369324
```

In [35]:

```
# Calculate confusion matrix
val_predictions_all = []
for classifier in svm_classifiers:
    val_predictions_all.append(classifier.decision_function(X_val_2d))
val_predictions_all = np.column_stack(val_predictions_all)
confusion_matrix = np.argmax(val_predictions_all, axis=1)

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix)
```

```
Confusion Matrix:
[ 1 13 11  8 10  2 15 18  6 18 14 10  3 15 15  3 15 15  6 15  7  6  5  3
 15  3 15 10  2 19 13  6 19  1  3 15  3 10  7 18  7  7  3  2  3  7 16 12
  7  0  3 10 19  3  8  3 15  7 14  3  3 15  2  5  0  3 15 18 13 18  7  5
  3 11 10 19 16 15 15 12  2 13  2 18  3  7 11  4 15  4 17  5  7 12  6 15
  0 19 12 18  8  7  8 18  3  5  7  4  8 12  3  0  6  6  3  7  7 10 15  7
  8 18 10 15  8 15  5 15  3 14  3  6  3  6  3 16  7  6 10 15 18 12  3  2
 12  2 18 14 15 18 18 15 13 15 10 11  9  6  6 10 18  3 10 19 13 15 15  0
  3  8  8  7 18  7 15  6 15 15  2 11 14  5 18  3 15 13 12 12 15 12 15 10
  8  1 18 15  7 15  3  3  3  6  3  7 15  7 10  1  0  1  3 18  3  6 15  7
  6  6  1 18  3 12 12  8  7  0  3  3 15 18 10 10  3 15 12 15 11 15 18  6
  3  3  7 15 12  5  8 15  5 15  4 13 15  6 18 18  7  3  3 13  3  3 15 18
 10 15 11 12 19 15 15  3 14  5  2  8 11 15  1  3  6  5  6  7  3  4  3  3
 15  3  0 15 10  5  3  5  6  8 15 12 14  7 15  6 15 15  8  3  7 10 14 15
  3 15 15 18  3  3 15  4  1 15 15 10 18 18  7 15 15 15  2 15 15 15  3 15
 12  9  6  0 13  3 10 18 15 11  3  8  6 12  3 18 15  3  7 15 10 14  8 12
 15 15 15 15  5 15  8  0 15  5  3  2  0  6  5  7  0  3  8  2  3 17  2  0
  0  3  4 10 15 15  7 10 15 10 15  2  4  3 10 19  3  6 10  5  7  8 12  3
  7  5 13 14  3  0 15  0  3  7  7  0  7  5  3 19  7 10 15  7 17 15  4  7
 17  3  3 15  8 12 10  8 15 18  7 14 15  1  7  6 16  5 18  0  3 10  3 15
 10  5 13  5 15 15 15  3 18  8  7 18  6 15  3  2  5 13  7  0  3  3 15  4
  3  8 13  2  6  4  4  3 13 12 10 18  6 12 13  3 13 17  5 14 13 15 17  0
 15 14  7 15  6 15  6 12  7 12 12 15 15 12  5 10  3 19 19  3 11  7  3  8
  5  2 11 10 12  8  3 15  2  0  3 12  7  8 18 15  0  7  2 10 15 18  3  6
 15  1  0  7  0  0  0  3  0  3 15  7 14 12 12  2  6  3  2 15 13  3 13 15
 10  5 12  0  0 15  3  5 15  7  3 10 15  7  8  3  3  3  7  7  0  8 13 13
  3 15  6 11  6 15 18 14  7  2  3 16 18 18 15  9 10  3 15  4 15  3  5 15
  3  7 18  1  7  6  8 13 11 15 15  3  5  6 15  3  2 16 15  8 18  3 10 15
 15 11  9 19  3 15 18  7 15 10 18 15  3  8 15 11  1 10 12  8  3 10 13  6
  6  7  3  0 10  7 10  0 15 15 19  2 18  8 13 15  3 15  7 12  1  3  3  3
  3  5  3  3  8 12 15 10 16 14 15 12 12  6  1 15  3 12 18 12 15  5 15 12
  7 18  6  7  8 14  5 12 13 15  8 15  6  3 15 12  7  7  3  5  5 10 15  5
 15  6  7  5  1 12 18  8 13 18 19  3  8  9  3 15 10  3  6 14  6  6  3 18
  3  5  3  6  3 19  0 12  3 18 14 13  3  6 15 12  6 15 18  8 15 15 15 11
 15  2 19 15  3 15 10  3  6 15  0 10  6  6 15 18  3 15 18  4  3  4 10  1
 14 15 10 12  3  6  6  8  7  0 12 15 10  3  1 15  0  6 15  3 18  3  2  1
 15  5  7 12 13 15  6  6  5  0 15  5  3  3  7 10  8  4 15  3 15 15  3 11
 16 15 17 19 12 15  0  7  3 12  7 15  5 15  7  3 15 15  8  3  7  3 13 10
 15 16 18 19  2  7  6  8  6  0 15  2 19 12 13  6 18 15 16  2  8  5 12 15
  0  7  6 15 11  9 12 13  1  4  3  8  6 19  2 15 13 17  6  5  3  6 15 11
 13 13 11 15  1 10  8  4  7 15 10 15  7  0 15  4 12  1  3  2  8 18 12  6
  5  0 15  3  8 15  3  6 15  0  0 15  8 10 15 18  6 15 15 15  7  7 11 10
```

```
 5  0 11  5 11 15 18]
```

In [ ]: