In this tutorial consider the CIFAR dataset. Use the initial code given below and perform the classification using CNN and HoG based features.

In [ ]:
```
# Auto-setup when running on Google Colab
if 'google.colab' in str(get_ipython()):
    !pip install openml

# General imports
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import openml as oml
import tensorflow as tf
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: openml in /usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: liac-arff>=2.4.0 in /usr/local/lib/python3.10/dist-packages (from openml) (2.5.0
)
Requirement already satisfied: xmltodict in /usr/local/lib/python3.10/dist-packages (from openml) (0.13.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from openml) (2.27.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from openml) (1.2
.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from openml) (2.8.2)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from openml) (1.5.3)
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.10/dist-packages (from openml) (1.10.1)
Requirement already satisfied: numpy>=1.6.2 in /usr/local/lib/python3.10/dist-packages (from openml) (1.22.4)
Requirement already satisfied: minio in /usr/local/lib/python3.10/dist-packages (from openml) (7.1.15)
Requirement already satisfied: pyarrow in /usr/local/lib/python3.10/dist-packages (from openml) (9.0.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ope
nml) (2022.7.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil->openm
l) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.1
8->openml) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-lea
rn>=0.18->openml) (3.1.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from minio->openml) (2022.12
.7)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from minio->openml) (1.26.15
)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from reque
sts->openml) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->openml)
(3.4)
```

In [ ]:
```
# Uncomment the next line if you run on Colab
#!pip install --quiet openml
```

In [ ]:
```
%matplotlib inline
import openml as oml
import matplotlib.pyplot as plt
```

In [ ]:
```
# Download CIFAR data. Takes a while the first time.
# This version returns 3x32x32 resolution images.
# If you feel like it, repeat the exercises with the 96x96x3 resolution version by using ID 41103
cifar = oml.datasets.get_dataset(40926)
X, y, _, _ = cifar.get_data(target=cifar.default_target_attribute, dataset_format='array');
cifar_classes = {0: "airplane", 1: "automobile", 2: "bird", 3: "cat", 4: "deer",
                 5: "dog", 6: "frog", 7: "horse", 8: "ship", 9: "truck"}
```

In [ ]:
```
# The dataset (40926) is in a weird 3x32x32 format, we need to reshape and transpose
Xr = X.reshape((len(X),3,32,32)).transpose(0,2,3,1)
```

In [ ]:
```
Xr
```

Out[ ]:
```
array([[[[ 59.,  62.,  63.],
         [ 43.,  46.,  45.],
         [ 50.,  48.,  43.],
         ...,
         [158., 132., 108.],
         [152., 125., 102.],
         [148., 124., 103.]],

        [[ 16.,  20.,  20.],
         [  0.,   0.,   0.],
         [ 18.,   8.,   0.],
         ...,
         [123.,  88.,  55.],
         [119.,  83.,  50.],
         [122.,  87.,  57.]],

        [[ 25.,  24.,  21.],
         [ 16.,   7.,   0.],
```

```
   [ 49.,  27.,   8.],
   ...,
   [118.,  84.,  50.],
   [120.,  84.,  50.],
   [109.,  73.,  42.]],

  ...,

  [[208., 170.,  96.],
   [201., 153.,  34.],
   [198., 161.,  26.],
   ...,
   [160., 133.,  70.],
   [ 56.,  31.,   7.],
   [ 53.,  34.,  20.]],

  [[180., 139.,  96.],
   [173., 123.,  42.],
   [186., 144.,  30.],
   ...,
   [184., 148.,  94.],
   [ 97.,  62.,  34.],
   [ 83.,  53.,  34.]],

  [[177., 144., 116.],
   [168., 129.,  94.],
   [179., 142.,  87.],
   ...,
   [216., 184., 140.],
   [151., 118.,  84.],
   [123.,  92.,  72.]]],


 [[[154., 177., 187.],
   [126., 137., 136.],
   [105., 104.,  95.],
   ...,
   [ 91.,  95.,  71.],
   [ 87.,  90.,  71.],
   [ 79.,  81.,  70.]],

  [[140., 160., 169.],
   [145., 153., 154.],
   [125., 125., 118.],
   ...,
   [ 96.,  99.,  78.],
   [ 77.,  80.,  62.],
   [ 71.,  73.,  61.]],

  [[140., 155., 164.],
   [139., 146., 149.],
   [115., 115., 112.],
   ...,
   [ 79.,  82.,  64.],
   [ 68.,  70.,  55.],
   [ 67.,  69.,  55.]],

  ...,

  [[175., 167., 166.],
   [156., 154., 160.],
   [154., 160., 170.],
   ...,
   [ 42.,  34.,  36.],
   [ 61.,  53.,  57.],
   [ 93.,  83.,  91.]],

  [[165., 154., 128.],
   [156., 152., 130.],
   [159., 161., 142.],
   ...,
   [103.,  93.,  96.],
   [123., 114., 120.],
   [131., 121., 131.]],

  [[163., 148., 120.],
   [158., 148., 122.],
   [163., 156., 133.],
   ...,
   [143., 133., 139.],
   [143., 134., 142.],
   [143., 133., 144.]]],


 [[[255., 255., 255.],
   [253., 253., 253.],
   [253., 253., 253.],
   ...,
   [253., 253., 253.],
```

```
         [253., 253., 253.],
         [253., 253., 253.]],

        [[255., 255., 255.],
         [255., 255., 255.],
         [255., 255., 255.],
         ...,
         [255., 255., 255.],
         [255., 255., 255.],
         [255., 255., 255.]],

        [[255., 255., 255.],
         [254., 254., 254.],
         [254., 254., 254.],
         ...,
         [254., 254., 254.],
         [254., 254., 254.],
         [254., 254., 254.]],

        ...,

        [[113., 120., 112.],
         [111., 118., 111.],
         [105., 112., 106.],
         ...,
         [ 72.,  81.,  80.],
         [ 72.,  80.,  79.],
         [ 72.,  80.,  79.]],

        [[111., 118., 110.],
         [104., 111., 104.],
         [ 99., 106.,  98.],
         ...,
         [ 68.,  75.,  73.],
         [ 70.,  76.,  75.],
         [ 78.,  84.,  82.]],

        [[106., 113., 105.],
         [ 99., 106.,  98.],
         [ 95., 102.,  94.],
         ...,
         [ 78.,  85.,  83.],
         [ 79.,  85.,  83.],
         [ 80.,  86.,  84.]]],


       ...,


       [[[ 20.,  15.,  12.],
         [ 19.,  14.,  11.],
         [ 15.,  14.,  11.],
         ...,
         [ 10.,   9.,   7.],
         [ 12.,  11.,   9.],
         [ 13.,  12.,  10.]],

        [[ 21.,  16.,  13.],
         [ 20.,  16.,  13.],
         [ 18.,  17.,  12.],
         ...,
         [ 10.,   9.,   7.],
         [ 10.,   9.,   7.],
         [ 12.,  11.,   9.]],

        [[ 21.,  16.,  13.],
         [ 21.,  17.,  12.],
         [ 20.,  18.,  11.],
         ...,
         [ 12.,  11.,   9.],
         [ 12.,  11.,   9.],
         [ 13.,  12.,  10.]],

        ...,

        [[ 33.,  25.,  13.],
         [ 34.,  26.,  15.],
         [ 34.,  26.,  15.],
         ...,
         [ 28.,  25.,  52.],
         [ 29.,  25.,  58.],
         [ 23.,  20.,  42.]],

        [[ 33.,  25.,  14.],
         [ 34.,  26.,  15.],
         [ 34.,  26.,  15.],
         ...,
         [ 27.,  24.,  52.],
         [ 27.,  24.,  56.],
```

```
  [ 25.,  22.,  47.]],

 [[ 31.,  23.,  12.],
  [ 32.,  24.,  13.],
  [ 33.,  25.,  14.],
  ...,
  [ 24.,  23.,  50.],
  [ 26.,  23.,  53.],
  [ 25.,  20.,  47.]]],


[[[ 25.,  40.,  12.],
  [ 15.,  36.,   3.],
  [ 23.,  41.,  18.],
  ...,
  [ 61.,  82.,  78.],
  [ 92., 113., 112.],
  [ 75.,  89.,  92.]],

 [[ 12.,  25.,   6.],
  [ 20.,  37.,   7.],
  [ 24.,  36.,  15.],
  ...,
  [115., 134., 138.],
  [149., 168., 177.],
  [104., 117., 131.]],

 [[ 12.,  25.,  11.],
  [ 15.,  29.,   6.],
  [ 34.,  40.,  24.],
  ...,
  [154., 172., 182.],
  [157., 175., 192.],
  [116., 129., 151.]],

 ...,

 [[100., 129.,  81.],
  [103., 132.,  84.],
  [104., 134.,  86.],
  ...,
  [ 97., 128.,  84.],
  [ 98., 126.,  84.],
  [ 91., 121.,  79.]],

 [[103., 132.,  83.],
  [104., 131.,  83.],
  [107., 135.,  87.],
  ...,
  [101., 132.,  87.],
  [ 99., 127.,  84.],
  [ 92., 121.,  79.]],

 [[ 95., 126.,  78.],
  [ 95., 123.,  76.],
  [101., 128.,  81.],
  ...,
  [ 93., 124.,  80.],
  [ 95., 123.,  81.],
  [ 92., 120.,  80.]]],


[[[ 73.,  78.,  75.],
  [ 98., 103., 113.],
  [ 99., 106., 114.],
  ...,
  [135., 150., 152.],
  [135., 149., 154.],
  [203., 215., 223.]],

 [[ 69.,  73.,  70.],
  [ 84.,  89.,  97.],
  [ 68.,  75.,  81.],
  ...,
  [ 85.,  95.,  89.],
  [ 71.,  82.,  80.],
  [120., 133., 135.]],

 [[ 69.,  73.,  70.],
  [ 90.,  95., 100.],
  [ 62.,  71.,  74.],
  ...,
  [ 74.,  81.,  70.],
  [ 53.,  62.,  54.],
  [ 62.,  74.,  69.]],

 ...,

 [[123., 128.,  96.],
```

```
        [132., 132., 102.],
        [129., 128., 100.],
        ...,
        [108., 107.,  88.],
        [ 62.,  60.,  55.],
        [ 27.,  27.,  28.]],

       [[115., 121.,  91.],
        [123., 124.,  95.],
        [129., 126.,  99.],
        ...,
        [115., 116.,  94.],
        [ 66.,  65.,  59.],
        [ 27.,  27.,  27.]],

       [[116., 120.,  90.],
        [121., 122.,  94.],
        [129., 128., 101.],
        ...,
        [116., 115.,  94.],
        [ 68.,  65.,  58.],
        [ 27.,  26.,  26.]]]], dtype=float32)
```

In [ ]:
```python
# Take some random examples, reshape to a 32x32 image and plot
from random import randint
fig, axes = plt.subplots(1, 5,  figsize=(10, 5))
for i in range(5):
    n = randint(0,len(Xr))
    # The data is stored in a 3x32x32 format, so we need to transpose it
    axes[i].imshow(Xr[n]/255)
    axes[i].set_xlabel((cifar_classes[int(y[n])]))
    axes[i].set_xticks(()), axes[i].set_yticks(())
plt.show();
```

Task 1: CNN based Classification

- Split the data into 80% training and 20% validation sets
- Normalize the data to [0,1]
- Build a ConvNet with 3 convolutional layers interspersed with MaxPooling layers, and one dense layer.
  - Use at least 32 3x3 filters in the first layer and ReLU activation.
  - Otherwise, make rational design choices or experiment a bit to see what works.
- You should at least get 60% accuracy.
- For training, you can try 20-50 epochs, but feel free to explore this as well

Spliting the data into 80% training and 20% validation

In [ ]:
```python
from sklearn.model_selection import train_test_split

# Split the data into 80% training and 20% validation sets
X_train, X_val, y_train, y_val = train_test_split(Xr, y, test_size=0.2, random_state=42)
X_train.shape
```

Out[ ]:
```
(16000, 32, 32, 3)
```

In [ ]:
```python
y_train.shape
```

Out[ ]:
```
(16000,)
```

In [ ]:
```python
#normalizing the data
X_train = X_train / 255.0
X_val = X_val / 255.0
```

In [ ]:
```python
# Build the ConvNet model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

In [ ]:
```
# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [ ]:
```
# Train the model
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_val, y_val))

# Evaluate the model on the validation set
val_loss, val_acc = model.evaluate(X_val, y_val)
```

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/backend.py:5612: UserWarning: "`sparse_categorical_crossentropy`
received `from_logits=True`, but the `output` argument was produced by a Softmax activation and thus does not r
epresent logits. Was this intended?
  output, from_logits = _get_logits(
500/500 [==============================] - 5s 5ms/step - loss: 1.7534 - accuracy: 0.3464 - val_loss: 1.5242 - v
al_accuracy: 0.4540
Epoch 2/50
500/500 [==============================] - 2s 4ms/step - loss: 1.3907 - accuracy: 0.4971 - val_loss: 1.4140 - v
al_accuracy: 0.4927
Epoch 3/50
500/500 [==============================] - 2s 4ms/step - loss: 1.2357 - accuracy: 0.5603 - val_loss: 1.3377 - v
al_accuracy: 0.5185
Epoch 4/50
500/500 [==============================] - 2s 4ms/step - loss: 1.1289 - accuracy: 0.5969 - val_loss: 1.2053 - v
al_accuracy: 0.5720
Epoch 5/50
500/500 [==============================] - 3s 7ms/step - loss: 1.0311 - accuracy: 0.6363 - val_loss: 1.2326 - v
al_accuracy: 0.5690
Epoch 6/50
500/500 [==============================] - 2s 4ms/step - loss: 0.9418 - accuracy: 0.6641 - val_loss: 1.1241 - v
al_accuracy: 0.6093
Epoch 7/50
500/500 [==============================] - 2s 4ms/step - loss: 0.8618 - accuracy: 0.6958 - val_loss: 1.1473 - v
al_accuracy: 0.6090
Epoch 8/50
500/500 [==============================] - 2s 4ms/step - loss: 0.7907 - accuracy: 0.7179 - val_loss: 1.1277 - v
al_accuracy: 0.6295
Epoch 9/50
500/500 [==============================] - 2s 4ms/step - loss: 0.7125 - accuracy: 0.7470 - val_loss: 1.1831 - v
al_accuracy: 0.6175
Epoch 10/50
500/500 [==============================] - 3s 5ms/step - loss: 0.6410 - accuracy: 0.7731 - val_loss: 1.1764 - v
al_accuracy: 0.6320
Epoch 11/50
500/500 [==============================] - 2s 5ms/step - loss: 0.5587 - accuracy: 0.8075 - val_loss: 1.2536 - v
al_accuracy: 0.6233
Epoch 12/50
500/500 [==============================] - 2s 4ms/step - loss: 0.4978 - accuracy: 0.8253 - val_loss: 1.3208 - v
al_accuracy: 0.6165
Epoch 13/50
500/500 [==============================] - 2s 4ms/step - loss: 0.4419 - accuracy: 0.8464 - val_loss: 1.3979 - v
al_accuracy: 0.6168
Epoch 14/50
500/500 [==============================] - 2s 4ms/step - loss: 0.3806 - accuracy: 0.8686 - val_loss: 1.4438 - v
al_accuracy: 0.6235
Epoch 15/50
500/500 [==============================] - 3s 5ms/step - loss: 0.3323 - accuracy: 0.8823 - val_loss: 1.5844 - v
al_accuracy: 0.5985
Epoch 16/50
500/500 [==============================] - 3s 5ms/step - loss: 0.2744 - accuracy: 0.9057 - val_loss: 1.6014 - v
al_accuracy: 0.6227
Epoch 17/50
500/500 [==============================] - 2s 5ms/step - loss: 0.2325 - accuracy: 0.9209 - val_loss: 1.8256 - v
al_accuracy: 0.6045
Epoch 18/50
500/500 [==============================] - 2s 4ms/step - loss: 0.2132 - accuracy: 0.9256 - val_loss: 1.8051 - v
al_accuracy: 0.6190
Epoch 19/50
500/500 [==============================] - 4s 8ms/step - loss: 0.1715 - accuracy: 0.9397 - val_loss: 2.0009 - v
al_accuracy: 0.6145
Epoch 20/50
500/500 [==============================] - 5s 10ms/step - loss: 0.1422 - accuracy: 0.9534 - val_loss: 2.0633 -
val_accuracy: 0.6160
Epoch 21/50
500/500 [==============================] - 3s 7ms/step - loss: 0.1336 - accuracy: 0.9556 - val_loss: 2.1868 - v
al_accuracy: 0.6223
Epoch 22/50
500/500 [==============================] - 2s 4ms/step - loss: 0.1338 - accuracy: 0.9549 - val_loss: 2.1824 - v
al_accuracy: 0.6090
Epoch 23/50
500/500 [==============================] - 2s 5ms/step - loss: 0.1179 - accuracy: 0.9588 - val_loss: 2.3605 - v
al_accuracy: 0.6165
```

```
Epoch 24/50
500/500 [==============================] - 3s 5ms/step - loss: 0.1009 - accuracy: 0.9664 - val_loss: 2.5011 - v
al_accuracy: 0.6047
Epoch 25/50
500/500 [==============================] - 2s 5ms/step - loss: 0.1010 - accuracy: 0.9653 - val_loss: 2.5552 - v
al_accuracy: 0.6110
Epoch 26/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0906 - accuracy: 0.9692 - val_loss: 2.6882 - v
al_accuracy: 0.6137
Epoch 27/50
500/500 [==============================] - 2s 4ms/step - loss: 0.1114 - accuracy: 0.9632 - val_loss: 2.8709 - v
al_accuracy: 0.5993
Epoch 28/50
500/500 [==============================] - 3s 6ms/step - loss: 0.0921 - accuracy: 0.9684 - val_loss: 2.6450 - v
al_accuracy: 0.6105
Epoch 29/50
500/500 [==============================] - 3s 6ms/step - loss: 0.0697 - accuracy: 0.9774 - val_loss: 2.8983 - v
al_accuracy: 0.6008
Epoch 30/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0861 - accuracy: 0.9711 - val_loss: 3.1602 - v
al_accuracy: 0.5900
Epoch 31/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0769 - accuracy: 0.9735 - val_loss: 2.9133 - v
al_accuracy: 0.5985
Epoch 32/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0951 - accuracy: 0.9672 - val_loss: 3.0931 - v
al_accuracy: 0.6148
Epoch 33/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0701 - accuracy: 0.9753 - val_loss: 3.1975 - v
al_accuracy: 0.6115
Epoch 34/50
500/500 [==============================] - 3s 6ms/step - loss: 0.0942 - accuracy: 0.9678 - val_loss: 3.0016 - v
al_accuracy: 0.6053
Epoch 35/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0535 - accuracy: 0.9827 - val_loss: 3.1260 - v
al_accuracy: 0.6045
Epoch 36/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0596 - accuracy: 0.9803 - val_loss: 3.1107 - v
al_accuracy: 0.6120
Epoch 37/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0665 - accuracy: 0.9771 - val_loss: 3.4090 - v
al_accuracy: 0.6058
Epoch 38/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0743 - accuracy: 0.9743 - val_loss: 3.2444 - v
al_accuracy: 0.6170
Epoch 39/50
500/500 [==============================] - 3s 5ms/step - loss: 0.0607 - accuracy: 0.9804 - val_loss: 3.3561 - v
al_accuracy: 0.5950
Epoch 40/50
500/500 [==============================] - 3s 5ms/step - loss: 0.0804 - accuracy: 0.9728 - val_loss: 3.3980 - v
al_accuracy: 0.5925
Epoch 41/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0587 - accuracy: 0.9808 - val_loss: 3.3581 - v
al_accuracy: 0.6227
Epoch 42/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0706 - accuracy: 0.9772 - val_loss: 3.4136 - v
al_accuracy: 0.6053
Epoch 43/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0682 - accuracy: 0.9769 - val_loss: 3.4453 - v
al_accuracy: 0.6083
Epoch 44/50
500/500 [==============================] - 3s 5ms/step - loss: 0.0420 - accuracy: 0.9859 - val_loss: 3.6554 - v
al_accuracy: 0.6155
Epoch 45/50
500/500 [==============================] - 3s 5ms/step - loss: 0.0536 - accuracy: 0.9815 - val_loss: 4.1795 - v
al_accuracy: 0.5838
Epoch 46/50
500/500 [==============================] - 2s 4ms/step - loss: 0.0981 - accuracy: 0.9671 - val_loss: 3.5745 - v
al_accuracy: 0.6028
Epoch 47/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0578 - accuracy: 0.9805 - val_loss: 3.5718 - v
al_accuracy: 0.6080
Epoch 48/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0305 - accuracy: 0.9899 - val_loss: 3.7301 - v
al_accuracy: 0.6065
Epoch 49/50
500/500 [==============================] - 2s 5ms/step - loss: 0.0589 - accuracy: 0.9808 - val_loss: 3.7318 - v
al_accuracy: 0.6093
Epoch 50/50
500/500 [==============================] - 3s 6ms/step - loss: 0.0852 - accuracy: 0.9722 - val_loss: 3.6693 - v
al_accuracy: 0.6012
125/125 [==============================] - 0s 2ms/step - loss: 3.6693 - accuracy: 0.6012
```

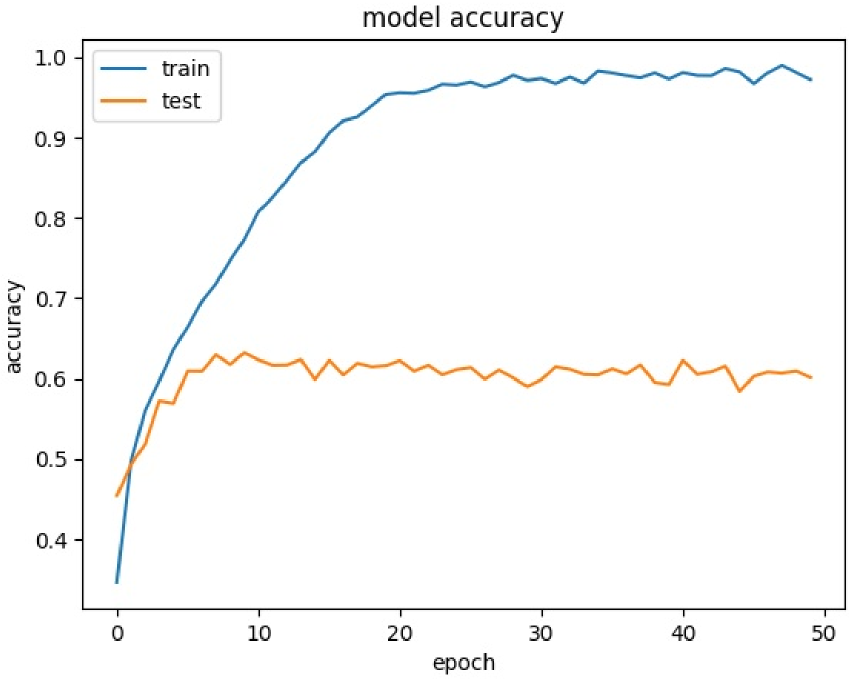
In [ ]:

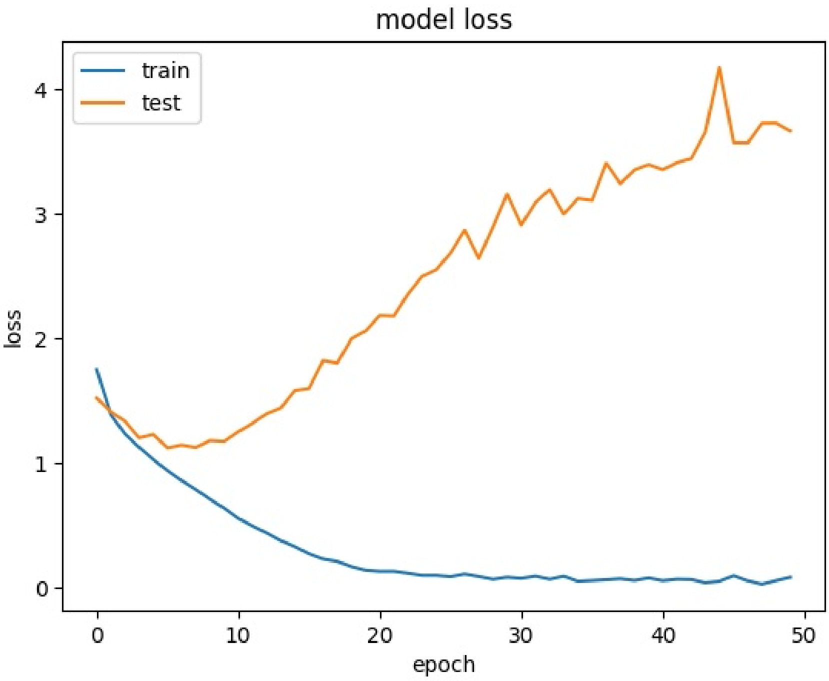
```
...
# list all data in history
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [ ]:
```
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





In [ ]:
```
print("Validation Loss:", val_loss)
print("Validation Accuracy:", val_acc*100)
```

```
Validation Loss: 3.669343948364258
Validation Accuracy: 60.12499928474426
```

In [ ]:
```
# Get the model's predictions on the validation set
y_val_pred = model.predict(X_val)
y_val_pred_labels = np.argmax(y_val_pred, axis=1)

# Calculate the accuracy by comparing the predicted labels with the true labels
val_accuracy = np.mean(y_val_pred_labels == y_val) * 100
```

```
# Print the validation accuracy
print("Validation Accuracy:", val_accuracy)
```

```
125/125 [==============================] - 0s 2ms/step
Validation Accuracy: 48.55
```

Task 2: HOG based Classification

- Use the same data split as mentioned above and extract the HOG features from the training and testing split and classify the HOG features using SVM classifier.
- Compare the accuracy with CNN features and explain which technique outperforms.

In [ ]:
```
from sklearn.model_selection import train_test_split

# Split the data into 80% training and 20% validation sets
X_train, X_val, y_train, y_val = train_test_split(Xr, y, test_size=0.2, random_state=42)
```

In [ ]:
```
from skimage.feature import hog

# Extract HOG features from the training set
X_train_hog = []
for image in X_train:
    features = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), multichannel=True)
    X_train_hog.append(features)
X_train_hog = np.array(X_train_hog)
```

```
<ipython-input-54-91c041c72d67>:6: FutureWarning: `multichannel` is a deprecated argument name for `hog`. It wi
ll be removed in version 1.0. Please use `channel_axis` instead.
  features = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), multichannel=True)
```

In [ ]:
```
# Extract HOG features from the validation set
X_val_hog = []
for image in X_val:
    features = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), multichannel=True)
    X_val_hog.append(features)
X_val_hog = np.array(X_val_hog)
```

```
<ipython-input-55-53234980e333>:4: FutureWarning: `multichannel` is a deprecated argument name for `hog`. It wi
ll be removed in version 1.0. Please use `channel_axis` instead.
  features = hog(image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), multichannel=True)
```

In [ ]:
```
from sklearn.svm import SVC

# Initialize and train the SVM classifier
svm_classifier = SVC()
svm_classifier.fit(X_train_hog, y_train)
accuracy_svm = svm_classifier.score(X_val_hog, y_val)
print("SVM Accuracy:", accuracy_svm)
```

```
SVM Accuracy: 0.5995
```

In [ ]:
```
results=pd.DataFrame({'Model':['CNN','SVM'],
                      'Accuracy Score':[accuracy_cnn,accuracy_svm]})
result_df=results.sort_values(by='Accuracy Score', ascending=False)
result_df=result_df.set_index('Model')
result_df
```

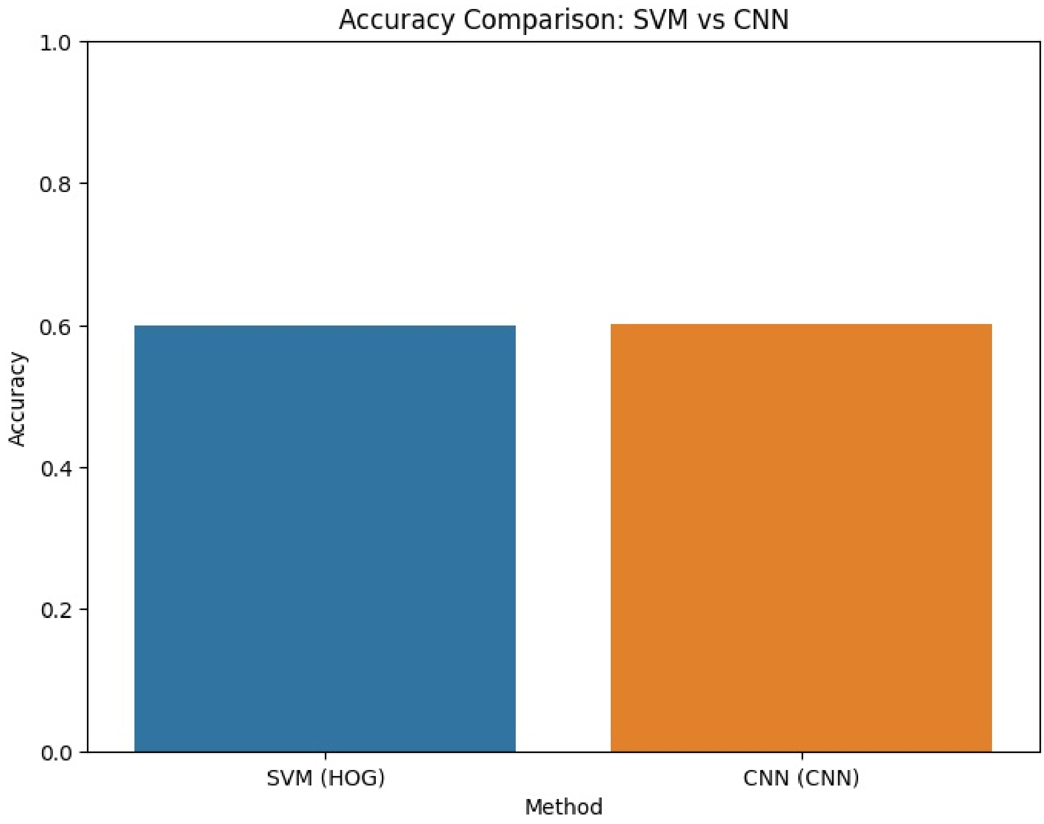
Out[ ]:

Model	Accuracy Score
CNN	0.60125
SVM	0.59950

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a list of method names and their corresponding accuracies
methods = ['SVM (HOG)', 'CNN (CNN)']
accuracies = [accuracy_svm, accuracy_cnn]

# Create a bar plot to visualize the accuracies
plt.figure(figsize=(8, 6))
sns.barplot(x=methods, y=accuracies)
plt.title('Accuracy Comparison: SVM vs CNN')
plt.xlabel('Method')
plt.ylabel('Accuracy')
plt.ylim([0, 1])  # Set the y-axis limit from 0 to 1
plt.show()
```


Accuracy Comparison: SVM vs CNN
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
SVM (HOG)
CNN (CNN)
Method