ODTÜ METU

***ME 536 INTELLIGENT MACHINE DESIGN COURSE***

***TERM PROJECT REPORT***

***FELINE DETECTOR***

*SUBMITTED BY:*

*REHA OĞUZ USLU*

*2308500*

## *AIM OF THE PROJECT*

Project aim is simply designing a feline detector. In our scenario, we have 4 kind of feline such as tiger, lion, leopard and cheetah. Aim is to detect that new input is one of these 4 kind or another kind of feline or completely another thing. Novelty detection is done by setting threshold on similarities.

$Max.Similarity > 50\%$ $\quad \rightarrow\rightarrow\rightarrow$ $It\ is\ not\ new.$

$50\% > Max.Similarity > 30\ \%$ $\quad \rightarrow\rightarrow\rightarrow$ $It\ is\ new\ kind\ of\ feline. It\ belong\ here.$

$Max.Similarity < 30\%$ $\quad \rightarrow\rightarrow\rightarrow$ $Completely\ new. Not\ belongs\ here.$

Three Topic Used in Project

- PCA
- ANN, CNN
- Image Processing (Specified Filters/Kernels)

### *DATA SPECIFICATIONS*

- 700 training images
- 300 test images
- 4 group such as leopard, lion, tiger, cheetah.
- All images are 64x64
- Single test image is inside the test_image folder.

### *THEORY AND METHODOLGY*

All steps are clearly explained below,

1. Required libraries are imported.

**Importing Required Libraries**

In [1]:
```
import numpy as np
import pandas as pd
import sklearn
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import initializers
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
import cv2
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

```
2023-01-28 19:26:52.814861: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorF
low binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following
CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

2. Training and Test data is imported by ImageDataGenerator. Pixels values are divided to 255 in order to compress between 0 and 1. In dataset, we have 700 images belongs to each class.

In [2]:
```
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True)
training_set = train_datagen.flow_from_directory(
    'training_dataset',
    target_size = (64,64),
    batch_size = 1,
    class_mode = 'categorical')


```

```
Found 2800 images belonging to 4 classes.
```

3. For CNN and ANN,  keras library by TensorFlow is used. Firstly, CNN is installed and added 2 convolution and pooling layer.

## *CNN*

### Installing CNN

- **Firstly convolution and maxpooling is done by convolutional neural network.**
- **Keras is used for CNN from TensorFlow.**

In [4]:
```
cnn = tf.keras.models.Sequential()
```

```
2023-01-28 19:27:11.520837: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorF
low binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following
CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

**First Convolution Layer**

In [5]:
```
d(tf.keras.layers.Conv2D(filters=16,kernel_size=3,activation = 'relu',input_shape=[64,64,3]
```

**First Pooling Layer**

In [6]:
```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

**Second Layer For CNN**

In [7]:
```
cnn.add(tf.keras.layers.Conv2D(filters=8,kernel_size=3,activation = 'relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

4. Now we have convolved and pooled data. Take data from CNN by below code.

In [8]:
```
data = cnn.predict(training_set)
```
```
2800/2800 [==============================] - 9s 3ms/step
```

- **Check dimensions of data**
  - **Expect to see 4D tensor**

In [9]:
```
data.shape
```
Out[9]:
```
(2800, 14, 14, 8)
```

5. Data is 4D tensor. First dimension represents number of images. Second and third dimensions represent width and height of images. Last dimension represents channels. Initially, our images' sizes are 64x64, however it reduces to 14x14 due to convolution and pooling.
6. Now, it is time to try 2 basic kernels/filters on our images. First kernel is 3x3 and it is edge detector from ME 536 classes. Below code is for creating kernel and performing on our data.

Edge Detection:

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\boxed{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Filter should be reshaped as 4D tensor.

## *FILTERING*

### Creating Filter/Kernel

In [10]:
```
kernel = tf.constant([
    [0, -1,  0],
    [-1, 4, -1],
    [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0],[0, -1,  0],[-1, 4, -1], [0, -1,  0],
    [0, -1,  0],[-1, 4, -1], [0, -1,  0]

], dtype=tf.float32, name='kernel')
```

In [11]:
```
kernel = tf.reshape(kernel, [3, 3, 1, 32], name='kernel')
```

- By using tf.nn library, filter is used on our data.

### Filtered Data With Specified Kernel

In [13]:
```
filtered_data= tf.nn.conv2d(
    data, kernel, [1,1,1,1], 'VALID',
    data_format='NHWC', dilations=[1, 1, 1, 1], name='kerneled'
)

```

- **Expect to see 12x12 image after convolution (remember we have images 14x14)**

In [14]:
```
filtered_data.shape
```

Out[14]:
```
TensorShape([2800, 12, 12, 32])
```

- Now second filter (5x5) is created. I create filter like circle, I expect to detect patterns on leopard with this filter. (Maybe it is stupid, I do not know, I will try ☺)

### Use 5x5 Kernel For This Time

In [15]:
```
kernel2 = tf.constant([
    [1, 1, 1, 1,1],
    [1,-1,-1,-1,1],
    [1,-1,-1,-1,1],
    [1,-1,-1,-1,1],
    [1, 1, 1, 1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],
    [1,1,1,1,1],[1,-1,-1,-1,1],[1, -1, -1,-1,1],[1,-1,-1,-1,1],[1,1,1,1,1],


], dtype=tf.float32, name='kernel')
```

In [16]:
```
kernel2 = tf.reshape(kernel2, [5, 5, 1, 32], name='kernel')
```

In [17]:
```
filtered_data2= tf.nn.conv2d(
    filtered_data, kernel2, [1,1,1,1], 'VALID',
    data_format='NHWC', dilations=[1, 1, 1, 1], name='kerneled'
)
```

In [18]:
```
filtered_data2.shape
```

Out[18]:
```
TensorShape([2800, 8, 8, 32])
```

- Now, we have filtered data, next step is flattening. Again, keras is used for flattening.

### Flattening Data

In [19]:
```
flatter = tf.keras.models.Sequential()
```

In [20]:
```
flatter.add(tf.keras.layers.Flatten())
```

In [21]:
```
flatten_data = flatter.predict(filtered_data2)
```

```
88/88 [==============================] - 0s 2ms/step
```

- **We expect to see 2800 data point with 8x8x32 = 2048 dimensions.**

```
In [22]: flatten_data.shape
Out[22]: (2800, 2048)
```

- We have 2800 points with 2048 dimensions ☺ WOW. It looks like we need to perform PCA analysis.
- First step of PCA ➔ Zero mean data

### Zero Mean Data

```
In [23]: def ZeroMean(M):
             return M - M.mean(axis=1).reshape((M.shape[0],1))

In [24]: data_zm = ZeroMean(flatten_data.T)
```

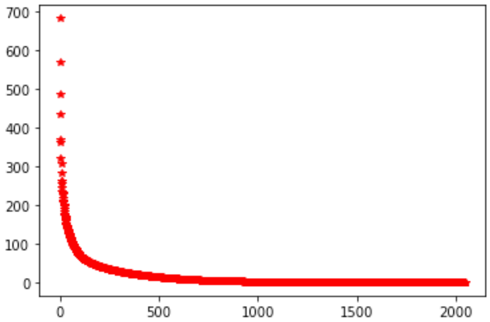
- SVD is performed via np.linald.svd

### SVD

```
In [25]: U,S,VT = np.linalg.svd(data_zm,full_matrices=False)
```

- Let's check sigma values graphically in order to estimate real rank.

```
In [25]: plt.plot(S,'r*')
Out[25]: [<matplotlib.lines.Line2D at 0x7fb910d1dd00>]
```




- Check first 15 sigma values

```
In [26]: print(S[0:15])

[681.514    568.7508   484.87863 433.18994 368.97595 362.8315   321.13245
 307.0515   282.2572   262.87512 260.6739   253.4559   243.00989 234.03564
 230.56721]
```

- Looks like real rank is 1, however it is hard to say something. Check calculated rank.

```
In [27]: np.linalg.matrix_rank(data_zm)
Out[27]: 1204
```

- 1024!!! WOW
- Let’s try energy method.

In [56]:
```
t = 0
for i in range(len(S)-1):
    t = t + (S[i])**2
```

In [29]:
```
k = 0
r = 0
for i in range(len(S)):
    k = k + (S[i])**2
    if k >= 0.9*t :
        break
    r = r + 1

```

In [30]:
```
print(f'Total energy:{t}\nRank Energy:{k}\nReal Rank:{r}')
```

```
Total energy:3987717.9350517844
Rank Energy:3589100.8272624
Real Rank:126
```

- We have different estimations for each method. Energy method estimation will be used. ☺

In [32]:
```
rank_mat = 126
```

In [33]:
```
data_w_pca = np.matmul(U[:,rank_mat].reshape(-1,1),VT[rank_mat,:].reshape(1,-1))
```

- After PCA analysis and Convolution is performed, data is ready to be embedded into ANN.

## ANN

In [36]:
```
ann = tf.keras.models.Sequential()
```

In [37]:
```
ann.add(tf.keras.layers.Dense(units=64, activation='relu'))
ann.add(tf.keras.layers.Dense(units=32, activation='relu'))
ann.add(tf.keras.layers.Dense(units=16, activation='relu'))
ann.add(tf.keras.layers.Dense(units=8, activation='relu'))

```

In [38]:
```
ann.add(tf.keras.layers.Dense(units=4, activation='softmax'))
```

In [39]:
```
ann.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

- Result array, y_set, is needed by ANN. It can be easily created by NumPy. Remember, y_set should be OneHotEncoded.

```
In [40]: a = np.zeros(700)
         b = 1*np.ones(700)
         c = 2*np.ones(700)
         d = 3*np.ones(700)

         y_set = np.concatenate((a,b,c,d))
```

```
In [41]: y_set_ =tf.one_hot(
             y_set,
             4,
             on_value=1,
             off_value=0,
             axis=-1,
             dtype=None,
             name=None)
```

- Train ANN with 100 epochs.

```
In [42]: ann.fit(data_w_pca, y_set_, batch_size = 32, epochs = 100)

Epoch 1/100
88/88 [==============================] - 1s 2ms/step - loss: 1.3866 - accuracy: 0.2482
Epoch 2/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3861 - accuracy: 0.2579
Epoch 3/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3860 - accuracy: 0.2511
Epoch 4/100
88/88 [==============================] - 0s 3ms/step - loss: 1.3857 - accuracy: 0.2664
Epoch 5/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3854 - accuracy: 0.2671
Epoch 6/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3853 - accuracy: 0.2593
Epoch 7/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3855 - accuracy: 0.2621
Epoch 8/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3853 - accuracy: 0.2671
Epoch 9/100
88/88 [==============================] - 0s 2ms/step - loss: 1.3857 - accuracy: 0.2636
Epoch 10/100
```

- Final accuracy is 27.68% ☹
- Now, single test image should be prepared. All steps are similar to train data preprocessing.

### Preprocessing Single Test Image

```
In [45]: test_image = train_datagen.flow_from_directory(
             'test_image',
             target_size = (64,64),
             batch_size = 1,
             class_mode = 'categorical')

Found 1 images belonging to 1 classes.
```

```
In [46]: test_image = cnn.predict(test_image)

1/1 [==============================] - 0s 59ms/step
```

```
In [47]: test_image.shape

Out[47]: (1, 14, 14, 8)
```

```
In [48]: filtered_test_img = tf.nn.conv2d(
             test_image, kernel, [1,1,1,1], 'VALID',
             data_format='NHWC', dilations=[1, 1, 1, 1], name='kerneled'
         )
```

```
In [49]: filtered_test_img_fnl = tf.nn.conv2d(
             filtered_test_img, kernel2, [1,1,1,1], 'VALID',
             data_format='NHWC', dilations=[1, 1, 1, 1], name='kerneled'
         )
```

```
In [50]: filtered_test_img = flatter.predict(filtered_test_img_fnl)

1/1 [==============================] - 0s 21ms/step
```

```
In [51]: filtered_test_img = ZeroMean(filtered_test_img)
```

```
In [52]: filtered_test_img.shape
Out[52]: (1, 2048)
```

## *SINGLE IMAGE TEST AND RESULT*

- Let's test below image

**Result**

```
In [112]: result = ann.predict(filtered_test_img)
          training_set.class_indices

          print(cnn.summary())

1/1 [==============================] - 0s 21ms/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 16)        448

 max_pooling2d (MaxPooling2D  (None, 31, 31, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 8)         1160

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 8)        0
 2D)
```

```
=================================================================
Total params: 1,608
Trainable params: 1,608
Non-trainable params: 0
_________________________________________________________________
None
```

### Creating Tresholds For Nolvalty Detection

In [113]:

```
print(f"Tiger Similarity : {result[0][2]*100}% \nLion Similarity : {result[0][1]*100}% \nLeopard Similarity : {resu

```

```
Tiger Similarity : 23.25342893600464%
Lion Similarity : 21.3715165853500037%
Leopard Similarity : 18.6143770813941966%
Cheetah Similarity : 36.76068186759949%
```

### Result Printer

In [114]:

```
m = max(result[0])
if m*100<50 and m*100>30:
    print('NEW \nNEW KIND OF FELINE \n')
elif m*100<=30:
    print('NEW \nNOT FELINE\n')
else:
    print(f'NOT NEW')

    if m == result[0][0]:
        print('IT IS TIGER!')
    elif m == result[0][1]:
        print('IT IS LION!')
    elif m == result[0][0]:
        print('IT IS LEOPARD!')
    else:
        print('IT IS CHETAH!')
```

```
NEW
NEW KIND OF FELINE
```