Project Name: Classifying Dogs & Cats Images using SVM(Support Vector Machine)

TASK 03

Description: Implement a support vector machine (SVM) to classify images of cats and dogs from the Kaggle dataset.

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

```
# Convolutional Neural Network

# Importing the libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
tf.__version__
```

```
'2.2.0'
```

```
# Part 1 - Data Preprocessing

# Preprocessing the Training set
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

```
training_set = train_datagen.flow_from_directory('Datasets/train',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

# Preprocessing the Test set
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('Datasets/test',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

```
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.regularizers import l2
```

```
# Part 2 - Building the CNN
# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Step 1 - Convolution
cnn.add(tf.keras.layers.Conv2D(filters=32,padding="same",kernel_size=3, activation='relu', strides=2, inp

# Step 2 - Pooling
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32,padding='same',kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

# Step 3 - Flattening
cnn.add(tf.keras.layers.Flatten())

# Step 4 - Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# Step 5 - Output Layer
#cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
## For Binary Classification
cnn.add(Dense(1, kernel_regularizer=tf.keras.regularizers.l2(0.01),activation
            ='linear'))
```

```
## for mulitclassification
cnn.add(Dense(4, kernel_regularizer=tf.keras.regularizers.l2(0.01),activation
            ='softmax'))
cnn.compile(optimizer = 'adam', loss = 'squared_hinge', metrics = ['accuracy'])
```

```
cnn.summary()
```

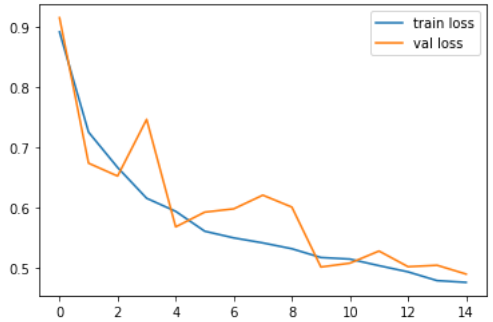
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 272,545
Trainable params: 272,545
```

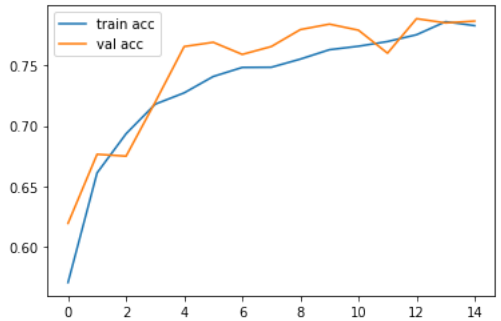
```
Non-trainable params: 0
_________________________________________________________________
```

```
# Part 3 - Training the CNN

# Compiling the CNN
cnn.compile(optimizer = 'adam', loss = 'hinge', metrics = ['accuracy'])

# Training the CNN on the Training set and evaluating it on the Test set
r=cnn.fit(x = training_set, validation_data = test_set, epochs = 15)
```

```
Epoch 1/15
250/250 [==============================] - 15s 58ms/step - loss: 0.8911 - accuracy: 0.5706 - val_loss
Epoch 2/15
250/250 [==============================] - 14s 56ms/step - loss: 0.7251 - accuracy: 0.6611 - val_loss
Epoch 3/15
250/250 [==============================] - 14s 56ms/step - loss: 0.6664 - accuracy: 0.6935 - val_loss
Epoch 4/15
250/250 [==============================] - 14s 56ms/step - loss: 0.6154 - accuracy: 0.7179 - val_loss
Epoch 5/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5937 - accuracy: 0.7272 - val_loss
Epoch 6/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5608 - accuracy: 0.7408 - val_loss
Epoch 7/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5497 - accuracy: 0.7483 - val_loss
Epoch 8/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5414 - accuracy: 0.7484 - val_loss
Epoch 9/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5317 - accuracy: 0.7551 - val_loss
Epoch 10/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5172 - accuracy: 0.7629 - val_loss
Epoch 11/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5146 - accuracy: 0.7657 - val_loss
Epoch 12/15
250/250 [==============================] - 14s 56ms/step - loss: 0.5037 - accuracy: 0.7696 - val_loss
Epoch 13/15
250/250 [==============================] - 14s 55ms/step - loss: 0.4934 - accuracy: 0.7753 - val_loss
Epoch 14/15
250/250 [==============================] - 14s 56ms/step - loss: 0.4789 - accuracy: 0.7860 - val_loss
Epoch 15/15
250/250 [==============================] - 14s 56ms/step - loss: 0.4760 - accuracy: 0.7828 - val_loss
```

```
# plot the loss
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```






```
<Figure size 432x288 with 0 Axes>
```

```
# save it as a h5 file


from tensorflow.keras.models import load_model

cnn.save('model_rcat_dog.h5')
```

```
from tensorflow.keras.models import load_model

# load model
model = load_model('model_rcat_dog.h5')
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
```

```
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

```
# Part 4 - Making a single prediction

import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('Datasets/test/dogs/dog.4015.jpg', target_size = (64,64))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
```

```
result
```

```
array([[2.13195]], dtype=float32)
```

```
# Part 4 - Making a single prediction

import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('Datasets/test/cats/cat.4017.jpg', target_size = (64,64))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
```

```
result
```

```
array([[-0.4654408]], dtype=float32)
```

```
if result[0]<0:
    print("The image classified is cat")
else:
    print("The image classified is dog")
```

```
The image classified is dog
```