Assiya Karatay karatay@bu.edu 8572947028

CS767 Machine Learning project

Image classification

Data Preprocessing

Import libraries

In [3]:
```
# base libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random

import os
from time import time
from zipfile import ZipFile
from sklearn.model_selection import train_test_split
# need to create new API token (the file kaggle.json) of Kaggle account and locate it in the same folder as the code
from kaggle.api.kaggle_api_extended import KaggleApi

# for the model
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras import Input
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dropout,Flatten,Dense,Activation,BatchNormalization
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing import image
from keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.applications import VGG16
```

Check the accelerator

In [4]:

```
print(tf.__version__)
print(keras.__version__)
if 'COLAB_TPU_ADDR' in os.environ:
    print('Connected to TPU')
elif tf.test.gpu_device_name() != '':
    print('Connected to GPU ' + tf.test.gpu_device_name())
else:
    print('Neither connected to a TPU nor a GPU')

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
    print('Select the Runtime → "Change runtime type" menu to enable a GPU accelerator, ')
    print('and then re-execute this cell.')
else:
    print(gpu_info)
```

```
2.8.0
2.8.0
Connected to GPU /device:GPU:0
Tue Apr 12 16:02:54 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   56C    P0    27W /  70W |    264MiB / 15109MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Initialize the parameters

In [5]:

```
seed = 96
```

```
tf.random.set_seed(seed)
np.random.seed(seed)
os.environ["PYTHONHASHSEED"] = str(seed)
random.seed(seed)
```

In [6]:
```
# initialize the parameters
image_size = (128, 128)
batch_size = 32
val_split = 0.2
epoch=2
# set the path of the data
train_path = './train'
```

Extracting data

We use Kaggle API to download the dataset from Kaggle.

In [7]:
```
# ! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

In [8]:
```
# initialise the API
kag = KaggleApi()
kag.authenticate()

# downloading the files
kag.competition_download_files(competition='dogs-vs-cats', path='./')

# unzip the files
with ZipFile('dogs-vs-cats.zip', 'r') as z:
    z.extractall()
with ZipFile('train.zip', 'r') as z:
    z.extractall()
```

In [9]:
```
os.remove("test1.zip")
os.remove("dogs-vs-cats.zip")
os.remove("sampleSubmission.csv")
```

In [10]:
```
# dataframe of labels
df = pd.DataFrame({'image_name':os.listdir(train_path)})
```

```
df['label'] =df['image_name'].apply(lambda x: x.split('.')[0])
df.head()
```

Out[10]:

	image_name	label
0	dog.8792.jpg	dog
1	dog.5129.jpg	dog
2	cat.10191.jpg	cat
3	dog.11241.jpg	dog
4	cat.6074.jpg	cat

Splitting the data

- train_data
- validation_data
- test_data

In [11]:

```
train_val_data, test_data = train_test_split(df,
                                        test_size = 0.2,
                                        stratify = df["label"],
                                        random_state = seed)
nb_test_samples = test_data.shape[0]
print(train_val_data.shape[0], nb_test_samples)
print(f'Size of test set: {nb_test_samples}')
```

```
20000 5000
Size of test set: 5000
```

In [12]:

```
train_data, val_data = train_test_split(train_val_data,
                                        test_size = 0.2,
                                        stratify = train_val_data["label"],
                                        random_state = seed)

print(f'Size of train set: {train_data.shape[0]}\nSize of validation set: {val_data.shape[0]} values')
```

```
Size of train set: 16000
Size of validation set: 4000 values
```

Data Visualization

In [13]:
```
fig = plt.figure(1, figsize = (10, 10))
fig.suptitle("Training sample images ")

for i in range(12):

    plt.subplot(6, 6, i + 1)
    image = load_img(train_path + '/'+ df["image_name"][i])
    plt.imshow(image)
    plt.axis("off")

plt.tight_layout()
plt.show()
```

Data Augmentation

Create data augmentation object

In [14]:
```
training_datagen = ImageDataGenerator(
      rescale = 1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
```

```
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
validation_datagen = ImageDataGenerator(rescale=1. / 255)
```

Generate data sets

In [15]:

```
train_generator = training_datagen.flow_from_dataframe(
    dataframe = train_data,
    directory = train_path,
    x_col = "image_name",
    y_col = "label",
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True,
    seed=seed)
```

```
Found 16000 validated image filenames belonging to 2 classes.
```

In [16]:

```
val_generator = validation_datagen.flow_from_dataframe(
    dataframe = val_data,
    directory = train_path,
    x_col = "image_name",
    y_col = "label",
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True,
    seed=seed)
```

```
Found 4000 validated image filenames belonging to 2 classes.
```

In [17]:

```
test_generator = validation_datagen.flow_from_dataframe(
    dataframe = test_data,
    directory = train_path,
    x_col = "image_name",
    y_col = "label",
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False,
    seed=seed)
```

```
Found 5000 validated image filenames belonging to 2 classes.
```

Visualize the augmented data

In [18]:
```
fig = plt.figure(1, figsize = (10, 10))
fig.suptitle("Training sample images ")

for i in range(12):

    plt.subplot(6, 6, i + 1)
    image, label = train_generator.next()
     # display the image from the iterator
    plt.imshow(image[0])
    plt.axis("off")

plt.tight_layout()
plt.show()
```

Model

Define the learning curve

In [19]:
```
# Plot the learning curve
def learning_curve(history,filename):
    # history.history is a dictionary containing the loss and measurements.
```

```
    # Change it to DataFrame, then method plot() can be used to get the learning curves.
    df = pd.DataFrame(history.history)
    df.tail(1).to_csv(filename+'_metrics.csv')
    print('Training metrics: ')
    print(df.tail(1))
    df.plot(figsize=(8, 5))
    plt.grid(True)
    plt.gca().set_ylim(0, 1) # Set the vertical range to [0,1]
    plt.show()
```

Define callbacks

In [20]:
```
early_stop = EarlyStopping(patience=10)

lr_reduction = ReduceLROnPlateau(
    monitor='val_accuracy',
    patience=2,
    verbose=1,
    factor=0.5,
    min_lr=0.00001
)
mc = ModelCheckpoint(filepath = './best_model.h5',save_best_only=True,
                                                  verbose=0)
model_chkpt = ModelCheckpoint('save_at_{epoch}.h5')

callbacks = [
    early_stop,
    lr_reduction,
    mc
]
```

Build a model

Train the model

In [21]:
```
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
```

```
    # The second convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 63, 63, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 flatten (Flatten)           (None, 4608)              0
```

```
 dropout (Dropout)           (None, 4608)             0

 dense (Dense)               (None, 512)              2359808

 dense_1 (Dense)             (None, 1)                513

=================================================================
Total params: 2,620,481
Trainable params: 2,620,481
Non-trainable params: 0
_________________________________________________________________
```

In [22]:

```
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

In [23]:

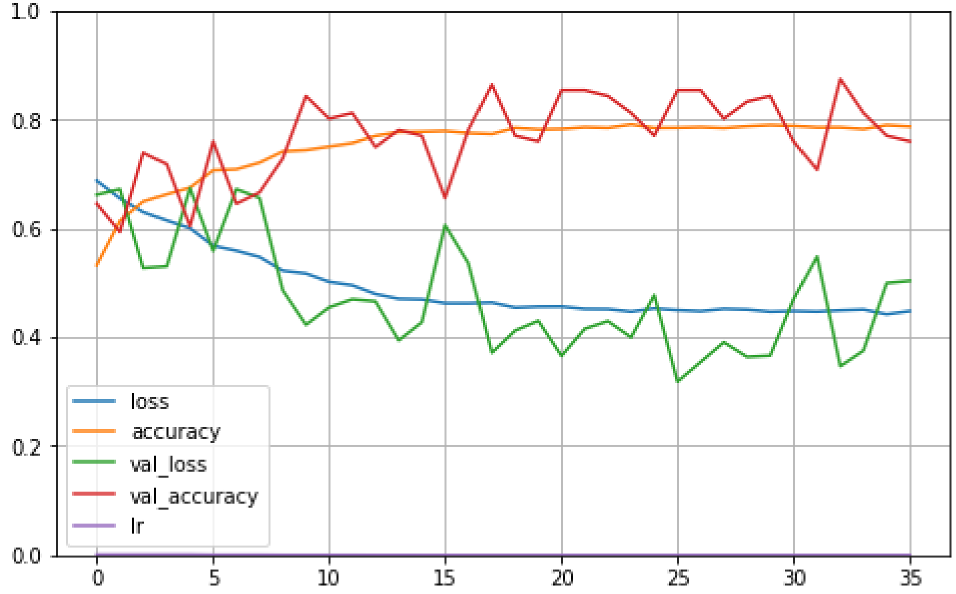
```
model_name = 'CNN'
print(model_name)
# training
print('training the model...')
start = time()
with tf.device("/GPU:0"): # use GPU-kernel
  hist = model.fit(train_generator,
          validation_data = val_generator,
          epochs=50,
          batch_size=batch_size,
          verbose = 1,
          validation_steps=3,
          callbacks=callbacks)
print('It takes ',round((time()-start)/60), 'mins to train the model.')

model.save('./final_' + model_name +'.h5')
test_loss, test_acc = model.evaluate(test_generator, verbose=0)
print('Test acc:', test_acc)
learning_curve(hist, model_name)
```

```
CNN
training the model...
Epoch 1/50
500/500 [==============================] - 116s 208ms/step - loss: 0.6881 - accuracy: 0.5326 - val_loss: 0.6624 - val_accuracy: 0.
6458 - lr: 0.0010
Epoch 2/50
500/500 [==============================] - 96s 192ms/step - loss: 0.6557 - accuracy: 0.6142 - val_loss: 0.6726 - val_accuracy: 0.5
938 - lr: 0.0010
Epoch 3/50
500/500 [==============================] - 96s 192ms/step - loss: 0.6304 - accuracy: 0.6499 - val_loss: 0.5274 - val_accuracy: 0.7
396 - lr: 0.0010
```

```
Epoch 4/50
500/500 [==============================] - 96s 192ms/step - loss: 0.6150 - accuracy: 0.6631 - val_loss: 0.5304 - val_accuracy: 0.7
188 - lr: 0.0010
Epoch 5/50
500/500 [==============================] - ETA: 0s - loss: 0.6007 - accuracy: 0.6754
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
500/500 [==============================] - 95s 191ms/step - loss: 0.6007 - accuracy: 0.6754 - val_loss: 0.6747 - val_accuracy: 0.6
042 - lr: 0.0010
Epoch 6/50
500/500 [==============================] - 95s 191ms/step - loss: 0.5683 - accuracy: 0.7074 - val_loss: 0.5587 - val_accuracy: 0.7
604 - lr: 5.0000e-04
Epoch 7/50
500/500 [==============================] - 96s 191ms/step - loss: 0.5592 - accuracy: 0.7092 - val_loss: 0.6727 - val_accuracy: 0.6
458 - lr: 5.0000e-04
Epoch 8/50
500/500 [==============================] - ETA: 0s - loss: 0.5477 - accuracy: 0.7216
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
500/500 [==============================] - 95s 191ms/step - loss: 0.5477 - accuracy: 0.7216 - val_loss: 0.6564 - val_accuracy: 0.6
667 - lr: 5.0000e-04
Epoch 9/50
500/500 [==============================] - 95s 190ms/step - loss: 0.5223 - accuracy: 0.7421 - val_loss: 0.4863 - val_accuracy: 0.7
292 - lr: 2.5000e-04
Epoch 10/50
500/500 [==============================] - 95s 191ms/step - loss: 0.5172 - accuracy: 0.7444 - val_loss: 0.4226 - val_accuracy: 0.8
438 - lr: 2.5000e-04
Epoch 11/50
500/500 [==============================] - 95s 190ms/step - loss: 0.5018 - accuracy: 0.7509 - val_loss: 0.4548 - val_accuracy: 0.8
021 - lr: 2.5000e-04
Epoch 12/50
500/500 [==============================] - ETA: 0s - loss: 0.4955 - accuracy: 0.7571
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
500/500 [==============================] - 95s 190ms/step - loss: 0.4955 - accuracy: 0.7571 - val_loss: 0.4698 - val_accuracy: 0.8
125 - lr: 2.5000e-04
Epoch 13/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4794 - accuracy: 0.7706 - val_loss: 0.4659 - val_accuracy: 0.7
500 - lr: 1.2500e-04
Epoch 14/50
500/500 [==============================] - ETA: 0s - loss: 0.4705 - accuracy: 0.7776
Epoch 14: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
500/500 [==============================] - 95s 190ms/step - loss: 0.4705 - accuracy: 0.7776 - val_loss: 0.3939 - val_accuracy: 0.7
812 - lr: 1.2500e-04
Epoch 15/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4696 - accuracy: 0.7782 - val_loss: 0.4275 - val_accuracy: 0.7
708 - lr: 6.2500e-05
Epoch 16/50
500/500 [==============================] - ETA: 0s - loss: 0.4624 - accuracy: 0.7794
Epoch 16: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
```

```
500/500 [==============================] - 95s 189ms/step - loss: 0.4624 - accuracy: 0.7794 - val_loss: 0.6064 - val_accuracy: 0.6
562 - lr: 6.2500e-05
Epoch 17/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4624 - accuracy: 0.7756 - val_loss: 0.5364 - val_accuracy: 0.7
812 - lr: 3.1250e-05
Epoch 18/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4636 - accuracy: 0.7739 - val_loss: 0.3713 - val_accuracy: 0.8
646 - lr: 3.1250e-05
Epoch 19/50
500/500 [==============================] - 94s 189ms/step - loss: 0.4545 - accuracy: 0.7852 - val_loss: 0.4121 - val_accuracy: 0.7
708 - lr: 3.1250e-05
Epoch 20/50
500/500 [==============================] - ETA: 0s - loss: 0.4561 - accuracy: 0.7823
Epoch 20: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
500/500 [==============================] - 95s 190ms/step - loss: 0.4561 - accuracy: 0.7823 - val_loss: 0.4299 - val_accuracy: 0.7
604 - lr: 3.1250e-05
Epoch 21/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4564 - accuracy: 0.7831 - val_loss: 0.3655 - val_accuracy: 0.8
542 - lr: 1.5625e-05
Epoch 22/50
500/500 [==============================] - ETA: 0s - loss: 0.4520 - accuracy: 0.7863
Epoch 22: ReduceLROnPlateau reducing learning rate to 1e-05.
500/500 [==============================] - 95s 189ms/step - loss: 0.4520 - accuracy: 0.7863 - val_loss: 0.4153 - val_accuracy: 0.8
542 - lr: 1.5625e-05
Epoch 23/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4516 - accuracy: 0.7851 - val_loss: 0.4294 - val_accuracy: 0.8
438 - lr: 1.0000e-05
Epoch 24/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4470 - accuracy: 0.7914 - val_loss: 0.3997 - val_accuracy: 0.8
125 - lr: 1.0000e-05
Epoch 25/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4529 - accuracy: 0.7850 - val_loss: 0.4773 - val_accuracy: 0.7
708 - lr: 1.0000e-05
Epoch 26/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4497 - accuracy: 0.7853 - val_loss: 0.3181 - val_accuracy: 0.8
542 - lr: 1.0000e-05
Epoch 27/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4478 - accuracy: 0.7864 - val_loss: 0.3543 - val_accuracy: 0.8
542 - lr: 1.0000e-05
Epoch 28/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4521 - accuracy: 0.7847 - val_loss: 0.3906 - val_accuracy: 0.8
021 - lr: 1.0000e-05
Epoch 29/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4508 - accuracy: 0.7881 - val_loss: 0.3637 - val_accuracy: 0.8
333 - lr: 1.0000e-05
Epoch 30/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4469 - accuracy: 0.7903 - val_loss: 0.3661 - val_accuracy: 0.8
```

```
438 - lr: 1.0000e-05
Epoch 31/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4481 - accuracy: 0.7887 - val_loss: 0.4696 - val_accuracy: 0.7
604 - lr: 1.0000e-05
Epoch 32/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4470 - accuracy: 0.7860 - val_loss: 0.5485 - val_accuracy: 0.7
083 - lr: 1.0000e-05
Epoch 33/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4493 - accuracy: 0.7861 - val_loss: 0.3464 - val_accuracy: 0.8
750 - lr: 1.0000e-05
Epoch 34/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4509 - accuracy: 0.7829 - val_loss: 0.3753 - val_accuracy: 0.8
125 - lr: 1.0000e-05
Epoch 35/50
500/500 [==============================] - 95s 190ms/step - loss: 0.4421 - accuracy: 0.7902 - val_loss: 0.4995 - val_accuracy: 0.7
708 - lr: 1.0000e-05
Epoch 36/50
500/500 [==============================] - 95s 189ms/step - loss: 0.4480 - accuracy: 0.7876 - val_loss: 0.5037 - val_accuracy: 0.7
604 - lr: 1.0000e-05
It takes  57 mins to train the model.
Test acc: 0.8163999915122986
Training metrics:
        loss  accuracy  val_loss  val_accuracy       lr
35  0.448034  0.787625  0.503682      0.760417  0.00001
```



In [23]: