OESON TRAINING FINAL PROJECT: DEEP LEARING-

HUMAN OR CAR?

BY ANASS EL AQLI

TABLE OF CONTENTS

INTRODUCTION

An essential capability for a self-driving car for example would be to distinguish between humans and cars. That's why for my final project at Oeson, I chose to build an image classification model using a deep learning approach. — In this presentation, we will delve into image data collection and cleaning, the model creation, training, and evaluation.

DATA COLLECTION

- The data collection part was the most time consuming step. I had to collect images for a “human” folder and a “car” folder.

- For the “human” file, to make sure the model gets trained on a diverse range of people. I spent hours collecting images of people from different backgrounds, ages, professions, etc.

- Also collected images of humans doing different activities and showing different emotions, and tried to include pictures with different angles of the human body.

DATA COLLECTION

- For the “car” file, I tried to feed the model different types of cars; older cars, futuristic ones, sports cars, trucks, etc.

- And I took into consideration the different angles so I included diverse car angles to the dataset.

- I also supplied images of dirty cars, rusty cars, etc. for a more natural interpretations of how vehicles could also look in real life.

DATA CLEANING

- In the cleaning process, I first manually deleted the very small images (< 9 KB) because they tend to be very low quality.

- Then I used this script which iterates through each image class within the 'data' directory.

- For each image file, it checks if the file format is within the accepted types. Corrupted or incompatible images are identified and removed.

```
data_dir = 'data' #dataset file
image_exts = ['jpeg','jpg', 'bmp', 'png'] # accepted image types
```

```
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
            os.remove(image_path)
```

DATA LOADING

- Employed TensorFlow to construct an image dataset.
- Then, utilized Matplotlib for visualizing a sample batch.
- class labels: 1 represents human, 0 represents car.

```
data = tf.keras.utils.image_dataset_from_directory('data')
```

```
Found 1100 files belonging to 2 classes.
```

```
data_iterator = data.as_numpy_iterator()
```

```
batch = data_iterator.next()
```

```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx]) # So class 1 = human & class 0 = car
```

DATA SPLITTING

- Employed a lambda function to scale pixel values between 0 and 1. Which enhances uniformity and facilitates model convergence.
- Partitioned the dataset into training, validation, and testing sets.
- 70% for training, 20% for validation, and 10% for testing.
- This is vital for evaluating model performance and ensuring generalization.

```
#Data Scaling
data = data.map(lambda x,y: (x/255, y))
```

```
data.as_numpy_iterator().next()
```

```
[[0.10968616, 0.03206284, 0.02166245],
 [0.18708353, 0.12787607, 0.08466318],
```

```
# Spliting data
train_size = int(len(data)*.7) # train data
val_size = int(len(data)*.2) # validation data
test_size = int(len(data)*.1) # testing data
```

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

DEEP LEARNING MODEL BUILDING

- Built a deep learning model for image classification using TensorFlow's Keras API.
- Utilized a Sequential model structure to stack layers.
- Integrated Convolutional Neural Network (CNN) layers for feature extraction:
- Conv2D layers with varying filter sizes and activation functions.
- MaxPooling2D layers for down-sampling spatial dimensions.
- Flattened the output and added fully connected Dense layers for classification.
- Applied ReLU activation to enhance non-linearity.
- Implemented a Sigmoid activation in the final layer for binary classification.
- Utilized the Adam optimizer and Binary Crossentropy loss function during model compilation.

```
model = Sequential()
```

```
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

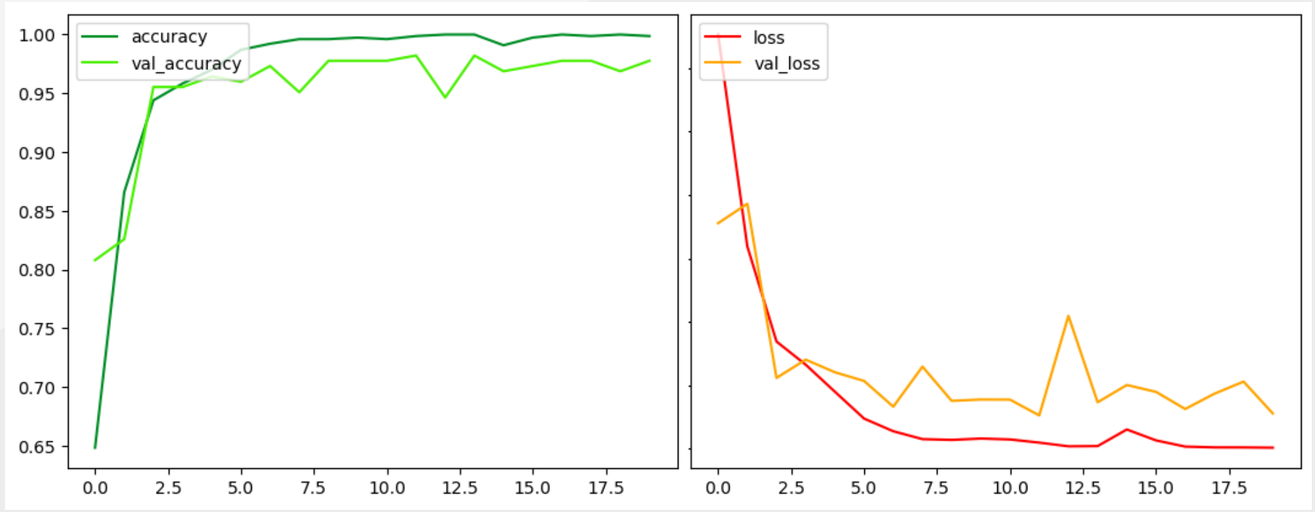
```
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

FITTING THE DL MODEL

- Established a TensorBoard callback for tracking training progress and storing log files.
- Conducted model training for 20 epochs on a dataset split into training and validation sets.
- Utilized the Adam optimizer and Binary Crossentropy loss function.
- Evaluated the model's accuracy and loss on both training and validation data after each epoch.
- Achieved notable accuracy improvements throughout epochs, reaching a validation accuracy of 97.77%.
- Demonstrated a quite steady convergence and model performance enhancement during the training process.

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

# Fitting
hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
```




```
Epoch 1/20
24/24 [==============================] - 67s 3s/step - loss: 0.6530 - accuracy: 0.6484 - val_loss: 0.3556 - val_accuracy: 0.8080
```

```
Epoch 20/20
24/24 [==============================] - 63s 3s/step - loss: 0.0017 - accuracy: 0.9987 - val_loss: 0.0557 - val_accuracy: 0.9777
```

TESTING

- Our model achieved an impressive 96.8% accuracy on the testing set, which promising and improvable for real-world advancement, in advanced domains like robotics and AI-driven vehicles.

```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

```
for batch in test.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)
    pre.update_state(y, yhat)
    re.update_state(y, yhat)
    acc.update_state(y, yhat)
```

```
print(f"""Precission: {pre.result()}
Recall: {re.result()}
Accuracy: {acc.result()}""")
```

```
Precission: 0.9599999785423279
Recall: 0.9795918464660645
Accuracy: 0.96875
```

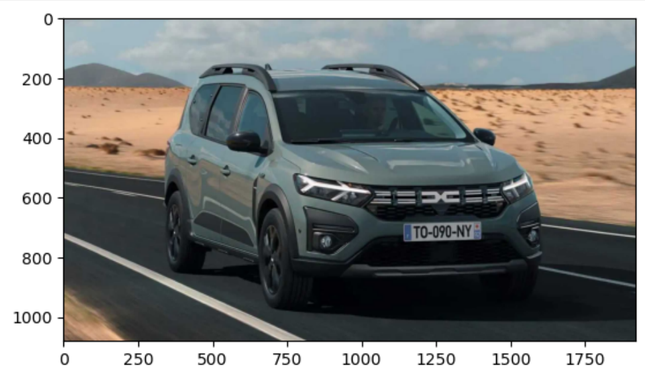
- But Let's test the model on some random images from the internet that were not part of the data and were not seen before by the model.

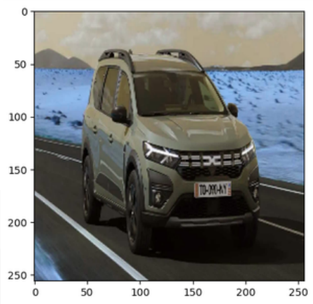
TESTING

- We tested our model on a random Dacia car image from the internet and a random Tesla car. And it classified them successfully.

```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```

```
yhat = model.predict(np.expand_dims(resize/255, 0))
```



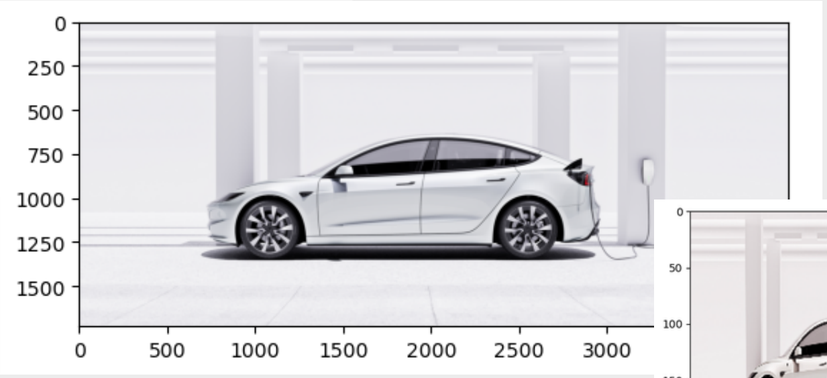


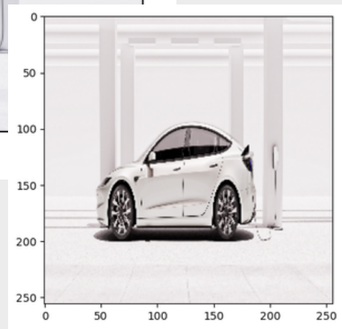
```
print(yhat)
```

```
[[3.721244e-12]]
```

```
if yhat > 0.5:
    print(f'Predicted class is human')
else:
    print(f'Predicted class is car')
```

```
Predicted class is car
```





```
yhat
```

```
array([[1.6433714e-05]], dtype=float32)
```

```
if yhat > 0.5:
    print(f'Predicted class is human')
else:
    print(f'Predicted class is car')
```

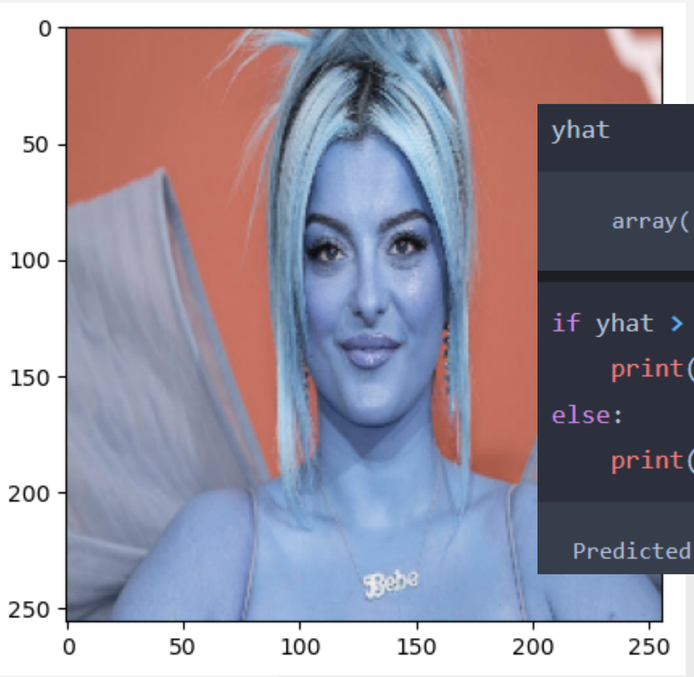
```
Predicted class is car
```

TESTING

- Also tested our model on a random image of Bebe Rexha from the internet and a random image of a group of people. And it classified them successfully!

```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```

```
yhat = model.predict(np.expand_dims(resize/255, 0))
```

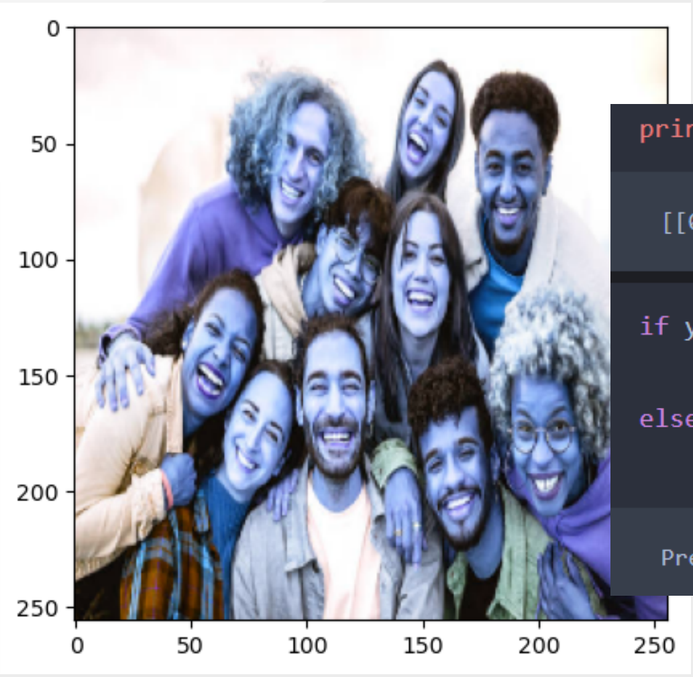



```
yhat
```

```
array([[0.98204774]], dtype=float32)
```

```
if yhat > 0.5:
    print(f'Predicted class is human')
else:
    print(f'Predicted class is car')
```

```
Predicted class is human
```




```
print(yhat)
```

```
[[0.99999577]]
```

```
if yhat > 0.5:
    print(f'Predicted class is human')
else:
    print(f'Predicted class is car')
```

```
Predicted class is human
```

Yeah, that's it, our model is doing pretty well based on the tests. It might need more tests and training on more types of cars and people. But from this first version, the model is promising!

THANK YOU