CNN with Distracted Driver Dataset Code

November 10, 2024

1 CNN with Distracted Driver Dataset Code

This Jupyter Notebook will have a focus on using the distracted driver dataset to detect whether a driver is falling asleep or being distracted. This model will serve as a warning detection system for the coding firmware.

1.0.1 Using Keras

Keras and Tensorflow are tools used to build and train machine learning models, especially deep learning models like neural networks. They are frameworks suited for deep learning tasks such as image recognition, natural language processing, and many more!

1.1 1 | Importing Libraries

Using a python installation, we import the following libraries / packages: - `numpy`: A library for numerical operations, providing support for large multi-dimensional arrays and matrices. - `pandas`: A data manipulation and analysis library that provides data structures like DataFrame for working with structured data. - `tensorflow`: An open-source framework for machine learning, enabling building and training of neural networks. - `os`: A module for interacting with the operating system, such as file and directory operations - `keras`: A high-level neural networks API, written in Python and capable of running on top of TensorFlow - `roboflow`: A package for accessing the Roboflow platform, which simplifies the process of training and deploying machine learning models - `sklearn`: Widely used open-source machine learning library for Python that provides simple and efficient tools for data mining and data analysis.

[1]:
```
%matplotlib inline

# importing all necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import os

# importing keras & roboflow
from tensorflow import keras
from roboflow import Roboflow
from PIL import Image
```

```
# using sklearn
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam

# keras imports
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, Input, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
```

```
2024-11-10 10:30:46.436407: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-11-10 10:30:46.459636: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1731252646.483022 1111736 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1731252646.490252 1111736 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2024-11-10 10:30:46.514877: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

1.2 2 | Accessing Dataset via Roboflow

Using Roboflow's API, we install our dataset of distracted driver images. We will be using this dataset to train our deep learning model and test it afterwards. Next, we use the validation is used to evaluate the deep learning model while its being trained.

[2]:
```
rf = Roboflow(api_key="jFtnSW2KoFxmJxA2kF50")
project = rf.workspace("yolov8-z7kip").
 ↪project("distracted-driver-detection-bvtnl")
version = project.version(2)
dataset = version.download("multiclass")
```

```
loading Roboflow workspace…
loading Roboflow project…
```

1.3 3 | Data Preprocessing

Firstly, we start off by using pandas. With our newly installed dataset, we access the `.csv` files from the test & train and directory by converting it into a dataframe. We then access the distracted column and store it in a variable.

[3]:
```
test_df = pd.read_csv('Distracted-Driver-Detection-2/test/_classes.csv')
distracted_test_col = np.array(test_df[' Distracted'].tolist())
```

[4]:
```
train_df = pd.read_csv('Distracted-Driver-Detection-2/train/_classes.csv')
distracted_train_col = np.array(train_df[' Distracted'].tolist())
```

1.3.1 Grayscaling & Normalizing

We design a function that takes in a array parameter. This function loops through the numpy arrays and checks whether it meets the requirements of the numpy array shape and check if the second index of the shape is 3.

Once the requirements are satisfied, we grayscale the numpy array and then noramlize it from 0 to 1 by dividing by 255. The new numpy arrays are appended to the `grayscale` array and then returned.

[5]:
```
def grayscale(rgb_arrays):
    grayscale = []

    for rgb_array in rgb_arrays:
        if len(rgb_array.shape) == 3 and rgb_array.shape[2] == 3:
            grayscale_image = 0.2989 * rgb_array[:, :, 0] + 0.5870 * rgb_array[:
 ↪, :, 1] + 0.1140 * rgb_array[:, :, 2]
            grayscale_image = grayscale_image / 255.0
            grayscale.append(grayscale_image)
        else:
            print("Not in RGB format")

    return np.array(grayscale)
```

1.3.2 Image Reading

Next, we move on to reading the image files. We access the directory of `images` and then store it in the `directory_path` variable. We set up a empty array that will store numpy arrays in the

variable `image_rgb_arrays`. We then read all the filenames in the `images` directory and then store them in the `filenames` variable.

[6]:
```
image_rgb_test_arrays = []
directory_test_path = "Distracted-Driver-Detection-2/test/images"
test_filenames = os.listdir(directory_test_path)

image_rgb_train_arrays = []
directory_train_path = "Distracted-Driver-Detection-2/train/images"
train_filenames = os.listdir(directory_train_path)
```

[7]:
```
for filename in test_filenames:
    image = Image.open("/home/chowdhuryj/madhacks/Distracted-Driver-Detection-2/
 ↪test/images/" + filename)
    image_array = np.array(image)
    image_rgb_test_arrays.append(image_array)

for filename in train_filenames:
    image = Image.open("/home/chowdhuryj/madhacks/Distracted-Driver-Detection-2/
 ↪train/images/" + filename)
    image_array = np.array(image)
    image_rgb_train_arrays.append(image_array)
```

1.3.3 Storing the New Numpy Arrays

Lastly, we store the grayscaled and noramlized numpy arrays in the `grayscale_normalized_images` variable. Next, we perform the following steps detailed below: 1. We rehape the numpy arrays to match the shape required by the CNN model so that we can prepare it for training the model 2. We then convert the data type of the numpy arrays to `float32` as it uses less memory and makes memory usage manageable

[8]:
```
grayscale_normalized_test_images = np.array(grayscale(image_rgb_test_arrays))
grayscale_normalized_train_images = np.array(grayscale(image_rgb_train_arrays))
```

[9]:
```
grayscale_normalized_train_images = grayscale_normalized_train_images.
 ↪reshape(-1, 640, 640, 1)
grayscale_normalized_test_images = grayscale_normalized_test_images.reshape(-1,␣
 ↪640, 640, 1)
```

[10]:
```
grayscale_normalized_train_images = grayscale_normalized_train_images.
 ↪astype("float32")
grayscale_normalized_test_images = grayscale_normalized_test_images.
 ↪astype("float32")
```

1.3.4 Data Augmentation

The final step of our data pre-processing is data augmentation. We end by performing data augmentation using `ImageDataGenerator`, which generates new, varied versions of the training images

in each epoch, which helps the model recognize patterns more effectively!

[11]:
```
# Create an instance of the ImageDataGenerator with augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=40,              # Random rotations (degrees)
    width_shift_range=0.2,          # Random horizontal shifts
    height_shift_range=0.2,         # Random vertical shifts
    shear_range=0.2,                # Random shear transformations
    zoom_range=0.2,                 # Random zoom
    horizontal_flip=True,           # Randomly flip images horizontally
    fill_mode='nearest'             # Strategy for filling missing pixels after␣
 ↪transformation
)
```

[12]:
```
datagen.fit(grayscale_normalized_train_images)
```

1.4 4 | Building the Convolutional Neural Network

Great! We're done with data pre-processing! Now, we move on to building our Convolutional Neural Network! We start off by initializing a Sequential model, which allows us to stack layers in order. We used a input later, convolutional, max pooling, flatten and dense layers as described in detail below:

1.4.1 Input, Convolutional & Max Pooling Layers

1. `Input Layer`: the input layer expects grayscale images which are 640 x 640
2. `First Convolutional Layer`: this layer applies 64 filters, each of the size 3 x 3 with a `ReLU` activation function
3. `First Max Pooling Layer`: this layer downsamples the feature maps, reducing the dimensions by taking the maximum value in each 3x3 window
4. `Second Convolutional Layer`: this layer applies 32 filters of the size `3x3` and uses `ReLU` for non-linearity
5. `Second Max Pooling Layer`: this layer downsamples the features maps, reducing data size and focuses on key patterns
6. `Flatten Layer`: this layer converts the 2D feature maps into a 1D array, preparing the data for the dense layers.

1.4.2 Dense & Output Layers

7. `First Dense Layer:` this is a dense layer with 128 neurons and ReLU activation, which learns high-level patterns
8. `Second Dense Layer`: this is a dense layer with 64 neurons, learning more abstracted patterns
9. `Third Dense Layer`: this is a dense layer with 32 neurons, which further abstracts the features
10. `Ouput Layer`: this is the final layer with 1 neuron and sigmoid activation function, which is ideal for binary classification

[13]:
```
# creating a model
model = Sequential()

# add input layer
model.add(Input(shape=(640, 640, 1)))

# adding the model layers
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
I0000 00:00:1731252665.629768 1111736 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13764 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:60:00.0, compute capability: 7.5
I0000 00:00:1731252665.640203 1111736 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13764 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:61:00.0, compute capability: 7.5
I0000 00:00:1731252665.649339 1111736 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:2 with 13764 MB memory:  -> device:
2, name: Tesla T4, pci bus id: 0000:da:00.0, compute capability: 7.5
I0000 00:00:1731252665.652124 1111736 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:3 with 13764 MB memory:  -> device:
3, name: Tesla T4, pci bus id: 0000:db:00.0, compute capability: 7.5
```

1.5 5 | Compliling the Model

Next, we compile the model, specify the optimization algorithm, the loss function and the metric for evaluating the model performance.

[14]:
```
#compile model using accuracy to measure model performance
model.compile(optimizer="adam", loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

1.6 6 | Training & Saving the Model

Lastly, we compile the model using the `fit()`, specifying the training data, validation data and the number of epochs.

[15]:
```
#train the model
model.fit(grayscale_normalized_train_images, distracted_train_col,
          validation_data=(grayscale_normalized_test_images,␣
 ↪distracted_test_col), epochs=10)
```

```
Epoch 1/10
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1731252669.833707 1112955 service.cc:148] XLA service 0x7fb36c0048f0
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1731252669.833737 1112955 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1731252669.833740 1112955 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1731252669.833742 1112955 service.cc:156]   StreamExecutor device
(2): Tesla T4, Compute Capability 7.5
I0000 00:00:1731252669.833744 1112955 service.cc:156]   StreamExecutor device
(3): Tesla T4, Compute Capability 7.5
2024-11-10 10:31:09.881496: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:268] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1731252670.121475 1112955 cuda_dnn.cc:529] Loaded cuDNN version
90300
2024-11-10 10:31:12.016009: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[32,64,638,638]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,1,640,640]{3,2,1,0},
f32[64,1,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}
2024-11-10 10:31:12.664452: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[32,32,210,210]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,64,212,212]{3,2,1,0},
f32[32,64,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}
2024-11-10 10:31:20.194206: E
external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm
eng0{} for conv (f32[32,64,3,3]{3,2,1,0}, u8[0]{0}) custom-
call(f32[32,64,212,212]{3,2,1,0}, f32[32,32,210,210]{3,2,1,0}),
window={size=3x3}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBackwardFilter", backend_config={"cudnn_conv_bac
kend_config":{"activation_mode":"kNone","conv_result_scale":1,"leakyrelu_alpha":
0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_id":"0"
```

,"wait_on_operation_queues":[]} is taking a while…
2024-11-10 10:31:20.783382: E
external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took
1.589355974s
Trying algorithm eng0{} for conv (f32[32,64,3,3]{3,2,1,0}, u8[0]{0}) custom-
call(f32[32,64,212,212]{3,2,1,0}, f32[32,32,210,210]{3,2,1,0}),
window={size=3x3}, dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBackwardFilter", backend_config={"cudnn_conv_bac
kend_config":{"activation_mode":"kNone","conv_result_scale":1,"leakyrelu_alpha":
0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_id":"0"
,"wait_on_operation_queues":[]} is taking a while…
I0000 00:00:1731252681.985009 1112955 device_compiler.h:188] Compiled cluster
using XLA! This line is logged at most once for the lifetime of the process.

27/28 **0s** 252ms/step -
accuracy: 0.6388 - loss: 1.4453

2024-11-10 10:31:30.216794: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[18,64,638,638]{3,2,1,0}, u8[0]{0}) custom-call(f32[18,1,640,640]{3,2,1,0},
f32[64,1,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}
2024-11-10 10:31:30.773075: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[18,32,210,210]{3,2,1,0}, u8[0]{0}) custom-call(f32[18,64,212,212]{3,2,1,0},
f32[32,64,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}

28/28 **0s** 564ms/step -
accuracy: 0.6392 - loss: 1.4270

2024-11-10 10:31:39.767773: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[32,64,638,638]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,1,640,640]{3,2,1,0},
f32[64,1,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_

```
id":"0","wait_on_operation_queues":[]}
2024-11-10 10:31:40.427898: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[32,32,210,210]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,64,212,212]{3,2,1,0},
f32[32,64,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}
2024-11-10 10:31:43.973179: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[28,64,638,638]{3,2,1,0}, u8[0]{0}) custom-call(f32[28,1,640,640]{3,2,1,0},
f32[64,1,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}
2024-11-10 10:31:44.560021: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:557]
Omitted potentially buggy algorithm eng14{k25=0} for conv
(f32[28,32,210,210]{3,2,1,0}, u8[0]{0}) custom-call(f32[28,64,212,212]{3,2,1,0},
f32[32,64,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward", backend_config={"cudnn_c
onv_backend_config":{"activation_mode":"kRelu","conv_result_scale":1,"leakyrelu_
alpha":0,"side_input_scale":0},"force_earliest_schedule":false,"operation_queue_
id":"0","wait_on_operation_queues":[]}

28/28            38s 898ms/step -
accuracy: 0.6396 - loss: 1.4099 - val_accuracy: 0.6859 - val_loss: 0.6521
Epoch 2/10
28/28            7s 260ms/step -
accuracy: 0.6406 - loss: 0.6595 - val_accuracy: 0.6859 - val_loss: 0.6351
Epoch 3/10
28/28            7s 261ms/step -
accuracy: 0.6643 - loss: 0.6399 - val_accuracy: 0.6859 - val_loss: 0.6353
Epoch 4/10
28/28            7s 261ms/step -
accuracy: 0.6513 - loss: 0.6314 - val_accuracy: 0.6731 - val_loss: 0.6364
Epoch 5/10
28/28            7s 263ms/step -
accuracy: 0.6840 - loss: 0.6030 - val_accuracy: 0.5833 - val_loss: 0.6575
Epoch 6/10
28/28            7s 264ms/step -
```

```
accuracy: 0.7339 - loss: 0.5473 - val_accuracy: 0.6474 - val_loss: 0.7024
Epoch 7/10
28/28             7s 263ms/step -
accuracy: 0.8204 - loss: 0.4252 - val_accuracy: 0.6090 - val_loss: 0.7207
Epoch 8/10
28/28             7s 263ms/step -
accuracy: 0.8682 - loss: 0.3370 - val_accuracy: 0.5962 - val_loss: 0.8625
Epoch 9/10
28/28             7s 263ms/step -
accuracy: 0.9196 - loss: 0.2131 - val_accuracy: 0.6026 - val_loss: 1.0381
Epoch 10/10
28/28             8s 266ms/step -
accuracy: 0.9470 - loss: 0.1910 - val_accuracy: 0.5962 - val_loss: 1.1836
```

[15]:
```
<keras.src.callbacks.history.History at 0x7fb4c802a400>
```

[16]:
```
model.save('distracted_driver_model.keras')
```