Distraction detection project-2

March 16, 2021

1 Imports

[1]:
```
import os
import pickle

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

from tqdm import tqdm

# Seaborn
import seaborn as sns
import seaborn as sns

# Sci-kit learn
from sklearn.datasets import load_files
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# Keras
from keras.utils import np_utils
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense, Input
from keras.models import Sequential
from keras.utils.vis_utils import plot_model
from keras.callbacks import ModelCheckpoint
from keras.utils import to_categorical
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import InceptionV3
from keras.optimizers import SGD
from keras.models import Model
```

```python
from PIL import ImageFile
```

2 Load the data

[2]:
```python
# Create path constants
DATA_DIR = r"state-farm-distracted-driver-detection/imgs"
TEST_DIR = os.path.join(DATA_DIR,"test")
TRAIN_DIR = os.path.join(DATA_DIR,"train")
MODEL_PATH = os.path.join(os.getcwd(),"basic_model_aug")
PICKLE_DIR = os.path.join(os.getcwd(),"pickle_files")
CSV_DIR = os.path.join(os.getcwd(),"csv_files")
```

[3]:
```python
# Create the specified directories (if it does not exist)
if not os.path.exists(TEST_DIR):
    print("Testing data does not exists")
if not os.path.exists(TRAIN_DIR):
    print("Training data does not exists")
if not os.path.exists(MODEL_PATH):
    print("Model path does not exists")
    os.makedirs(MODEL_PATH)
    print("Model path created")
if not os.path.exists(PICKLE_DIR):
    os.makedirs(PICKLE_DIR)
if not os.path.exists(CSV_DIR):
    os.makedirs(CSV_DIR)
```

3 Create data CSVs

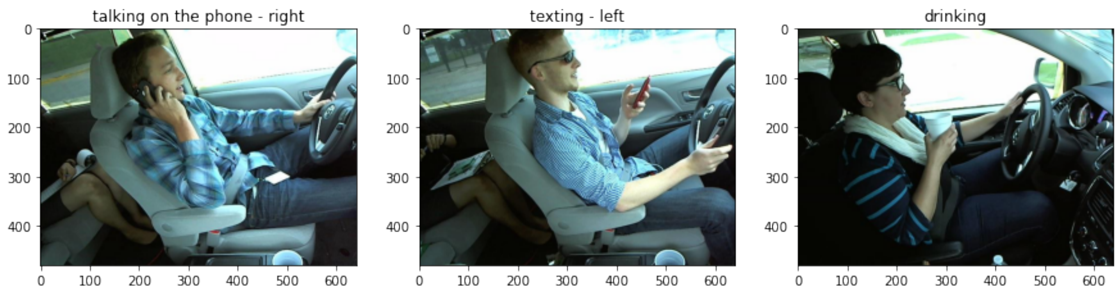
[4]:
```python
def create_csv(DATA_DIR, filename):
    # Get the classes
    class_names = os.listdir(DATA_DIR)
    data = list()
    # Check if the class data folder is found if not we are in test directory
    if(os.path.isdir(os.path.join(DATA_DIR, class_names[0]))):
        # iterate through the classes
        for class_name in class_names:
            file_names = os.listdir(os.path.join(DATA_DIR, class_name))
            for file in file_names:
                data.append({
                    "filename": os.path.join(DATA_DIR,class_name,file),
                    "classname": class_name
                })
    else:
        class_name = "test"
        file_names = os.listdir(DATA_DIR)
        for file in file_names:
```

```
            data.append(({
                "filename": os.path.join(DATA_DIR,file),
                "classname": class_name
            }))
    data = pd.DataFrame(data)
    data.to_csv(os.path.join(os.getcwd(), "csv_files", filename), index=False)
```

[5]:
```
# Create CSVs for train and test
create_csv(TRAIN_DIR,"train.csv")
create_csv(TEST_DIR,"test.csv")
# Load the train and test CSVs
data_train = pd.read_csv(os.path.join(os.getcwd(), "csv_files", "train.csv"))
data_test = pd.read_csv(os.path.join(os.getcwd(), "csv_files", "test.csv"))
```

3.1 Data visualization

[6]:
```
classes = {"c0": "safe driving",
           "c1": "texting - right",
           "c2": "talking on the phone - right",
           "c3": "texting - left",
           "c4": "talking on the phone - left",
           "c5": "operating the radio",
           "c6": "drinking",
           "c7": "reaching behind",
           "c8": "hair and makeup",
           "c9": "talking to passenger"
          }
# Take a random sample of the da a
df = data_train.sample(frac=1).reset_index(drop=True)
plt.figure(figsize=(15, 15))
for i, row in df.iterrows():
    img_path = row.values[0]
    img = image.load_img(img_path)
    ax = plt.subplot(int(f"23{i+1}"))
    ax.margins(0.05)
    ax.imshow(img)
    label = row.values[1]
    label = classes[label]
    ax.set_title(label)
    if i > 4:
        break
plt.savefig(os.path.join(MODEL_PATH, "data_visualization_basic.png"))
plt.show()
```



3.2 Data exploration

[7]:
```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22424 entries, 0 to 22423
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   filename   22424 non-null  object
 1   classname  22424 non-null  object
dtypes: object(2)
memory usage: 350.5+ KB
```

[8]:
```
data_train['classname'].value_counts()
```

[8]:
```
c0    2489
c3    2346
c4    2326
c6    2325
```

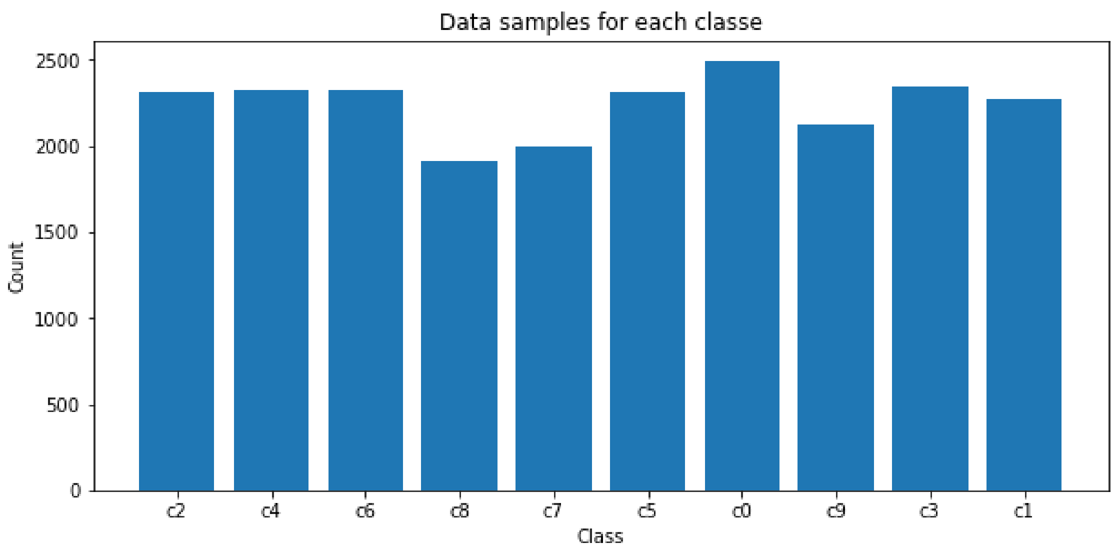
```
c2    2317
c5    2312
c1    2267
c9    2129
c7    2002
c8    1911
Name: classname, dtype: int64
```

[9]:
```python
data_train.describe()
```

[9]:

	filename	classname
count	22424	22424
unique	22424	10
top	state-farm-distracted-driver-detection/imgs\tr…	c0
freq	1	2489

[10]:
```python
# Create a bar chart for the data samples for each class
nf = data_train['classname'].value_counts(sort=False)
labels = data_train['classname'].value_counts(sort=False).index.tolist()
y = np.array(nf)
x = range(len(y))
# Create figure
fig = plt.figure(figsize=(10, 10))
ay = fig.add_subplot(211)
plt.xticks(x, labels)
ay.bar(x, y)
plt.title('Data samples for each classe')
plt.xlabel('Class')
plt.ylabel('Count')
plt.savefig(os.path.join(MODEL_PATH, "data_samples_per_class.png"))
plt.show()
```



We can see that the training dataset is equally balanced to a great extent and hence we need not do any downsampling of the data.

[11]:
```
data_test.head()
```

[11]:
```
                                            filename classname
0  state-farm-distracted-driver-detection/imgs\te…      test
1  state-farm-distracted-driver-detection/imgs\te…      test
2  state-farm-distracted-driver-detection/imgs\te…      test
3  state-farm-distracted-driver-detection/imgs\te…      test
4  state-farm-distracted-driver-detection/imgs\te…      test
```

[12]:
```
print(f"There are total {data_train.shape[0]} training samples")
print(f"There are total {data_test.shape[0]} testing samples")
```

```
There are total 22424 training samples
There are total 79726 testing samples
```

4 Data preprocessing

[13]:
```
# Create path constants
DATA_DIR = "state-farm-distracted-driver-detection\\imgs\\train\\"
images_df = pd.
 ↪read_csv("state-farm-distracted-driver-detection\\driver_imgs_list.csv")
print(images_df.head())

# Add the full path to the image name
for i, row in images_df.iterrows():
    row["img"] = DATA_DIR + row["classname"] + "\\" + row["img"]

# Count the number of sample in each class
images_df["subject"].value_counts()
```

```
  subject classname            img
0    p002        c0  img_44733.jpg
1    p002        c0  img_72999.jpg
2    p002        c0  img_25094.jpg
3    p002        c0  img_69092.jpg
4    p002        c0  img_92629.jpg
```

[13]:
```
p021    1237
p022    1233
p024    1226
p026    1196
p016    1078
```

```
p066    1034
p049    1011
p051     920
p014     876
p015     875
p035     848
p047     835
p081     823
p012     823
p064     820
p075     814
p061     809
p056     794
p050     790
p052     740
p002     725
p045     724
p039     651
p041     605
p042     591
p072     346
Name: subject, dtype: int64
```

4.1 Convert the labels to numerals

[14]:
```
labels_list = list(set(images_df['classname'].values.tolist()))
labels_id = {label_name:id for id, label_name in enumerate(labels_list)}
print(f"labels_id = {labels_id}")
images_df['classname'].replace(labels_id, inplace=True)
images_df['classname'] = images_df['classname'].apply(str)
```

```
labels_id = {'c0': 0, 'c1': 1, 'c3': 2, 'c7': 3, 'c9': 4, 'c5': 5, 'c8': 6,
'c6': 7, 'c4': 8, 'c2': 9}
```

4.2 Stratified sampling

[15]:
```
# Split the data into 80% train and 20% test (stratified)
test_df = images_df.loc[(images_df["subject"] == "p021") |␣
 ↪(images_df["subject"] == "p022") | \
                        (images_df["subject"] == "p024") |␣
 ↪(images_df["subject"] == "p026")]


mask = images_df["img"].isin(test_df["img"])
train_df = images_df.drop(images_df[mask].index)

train_df.drop("subject", axis=1, inplace=True)
```

```
test_df.drop("subject", axis=1, inplace=True)

print(f"images_df.shape = {images_df.shape}")
print(f"train_df.shape = {train_df.shape}")
print(f"test_df.shape = {test_df.shape}")
```

```
images_df.shape = (22424, 3)
train_df.shape = (17532, 2)
test_df.shape = (4892, 2)
```

```
C:\Users\asebaq\anaconda3\lib\site-packages\pandas\core\frame.py:4163:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

4.3 Data augmentation

[16]:
```
batch_size = 32
target_size = (64, 64)
# Create the training data generator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)



train_generator = train_datagen.flow_from_dataframe(train_df,
                                                    x_col="img",
                                                    y_col="classname",
                                                    target_size=target_size,
                                                    batch_size=batch_size,
                                                    class_mode="categorical")
```

```
Found 17532 validated image filenames belonging to 10 classes.
```

[17]:
```
print("Applying scaling, shear, zoom, and horizontal flip augmentation␣
 ↪techniques")
plt.figure(figsize=(15, 15))
for i in range(8):
    img, label = train_generator.next()
    ax = plt.subplot(int(f"44{i+1}"))
    ax.margins(0.05)
    ax.imshow(img[0])
plt.savefig(os.path.join(MODEL_PATH, "data_augmentation.png"))
plt.show()
```

```
Applying scaling, shear, zoom, and horizontal flip augmentation techniques
```

```
[18]: # Create the validation data generator
test_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = test_datagen.flow_from_dataframe(test_df,
                                                        x_col="img",
                                                        y_col="classname",
                                                        target_size=target_size,
                                                        batch_size=batch_size,
                                                        ␣
 ↪class_mode='categorical')
```

```
Found 4892 validated image filenames belonging to 10 classes.
```

5 Building the Model

```
[19]: input_shape = (64, 64, 3)
kernel_initializer = 'glorot_normal'
padding = 'same'
activation = 'relu'

model = Sequential()

model.add(Conv2D(filters=64, kernel_size=3, padding=padding,␣
 ↪activation=activation,
                 input_shape=input_shape,␣
 ↪kernel_initializer=kernel_initializer))
```

```
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=128, kernel_size=3, padding=padding,␣
↪activation=activation,
                 kernel_initializer=kernel_initializer))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=256, kernel_size=3, padding=padding,␣
↪activation=activation,
                 kernel_initializer=kernel_initializer))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=512, kernel_size=3, padding=padding,␣
↪activation=activation,
                 kernel_initializer=kernel_initializer))
model.add(MaxPooling2D(pool_size=2))

model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512, activation=activation,␣
↪kernel_initializer=kernel_initializer))
model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax',␣
↪kernel_initializer=kernel_initializer))


model.summary()
```

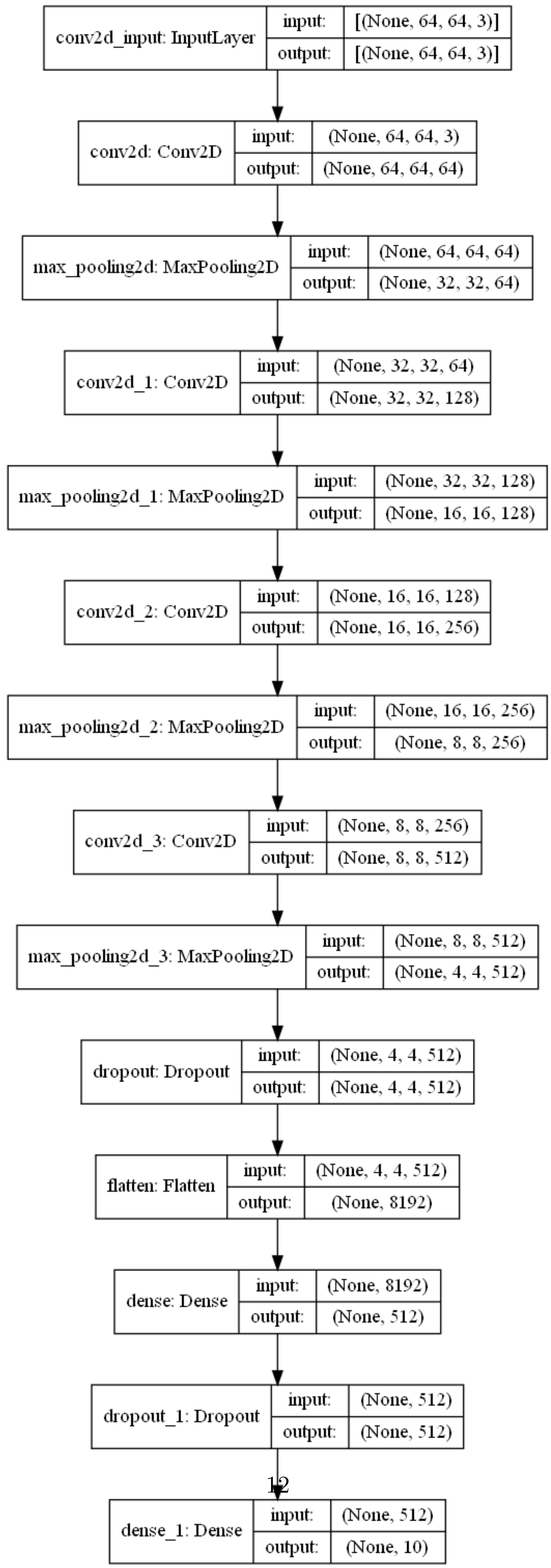
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 256)       295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 512)         1180160
_________________________________________________________________
```

```
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4194816
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 5,750,922
Trainable params: 5,750,922
Non-trainable params: 0
_________________________________________________________________
```

[20]:

```python
# Plot the model
plot_model(model, to_file=os.path.join(MODEL_PATH, "basic_distraction_model_aug.
 ↪png"), show_shapes=True, show_layer_names=True)
```

[20]:


conv2d_input: InputLayer
input: [(None, 64, 64, 3)]
output: [(None, 64, 64, 3)]
conv2d: Conv2D
input: (None, 64, 64, 3)
output: (None, 64, 64, 64)
max_pooling2d: MaxPooling2D
input: (None, 64, 64, 64)
output: (None, 32, 32, 64)
conv2d_1: Conv2D
input: (None, 32, 32, 64)
output: (None, 32, 32, 128)
max_pooling2d_1: MaxPooling2D
input: (None, 32, 32, 128)
output: (None, 16, 16, 128)
conv2d_2: Conv2D
input: (None, 16, 16, 128)
output: (None, 16, 16, 256)
max_pooling2d_2: MaxPooling2D
input: (None, 16, 16, 256)
output: (None, 8, 8, 256)
conv2d_3: Conv2D
input: (None, 8, 8, 256)
output: (None, 8, 8, 512)
max_pooling2d_3: MaxPooling2D
input: (None, 8, 8, 512)
output: (None, 4, 4, 512)
dropout: Dropout
input: (None, 4, 4, 512)
output: (None, 4, 4, 512)
flatten: Flatten
input: (None, 4, 4, 512)
output: (None, 8192)
dense: Dense
input: (None, 8192)
output: (None, 512)
dropout_1: Dropout
input: (None, 512)
output: (None, 512)
dense_1: Dense
input: (None, 512)
output: (None, 10)

[21]:
```
# Compile the model
model.compile(optimizer='sgd', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[22]:
```
steps_per_epoch = train_df.shape[0] / batch_size
validation_steps = test_df.shape[0] / batch_size
epochs = 10
# Save the model with the best accuracy during the training process
filepath = os.path.join(MODEL_PATH, "distraction-{epoch:02d}-{val_accuracy:.2f}.
 ↪hdf5")
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,␣
 ↪save_best_only=True, mode='max', period=1)
callbacks_list = [checkpoint]

# Start the training process
model_history = model.fit(train_generator,
                          steps_per_epoch=steps_per_epoch,
                          epochs=epochs,
                          validation_data=validation_generator,
                          validation_steps=validation_steps,
                          callbacks=callbacks_list)
```

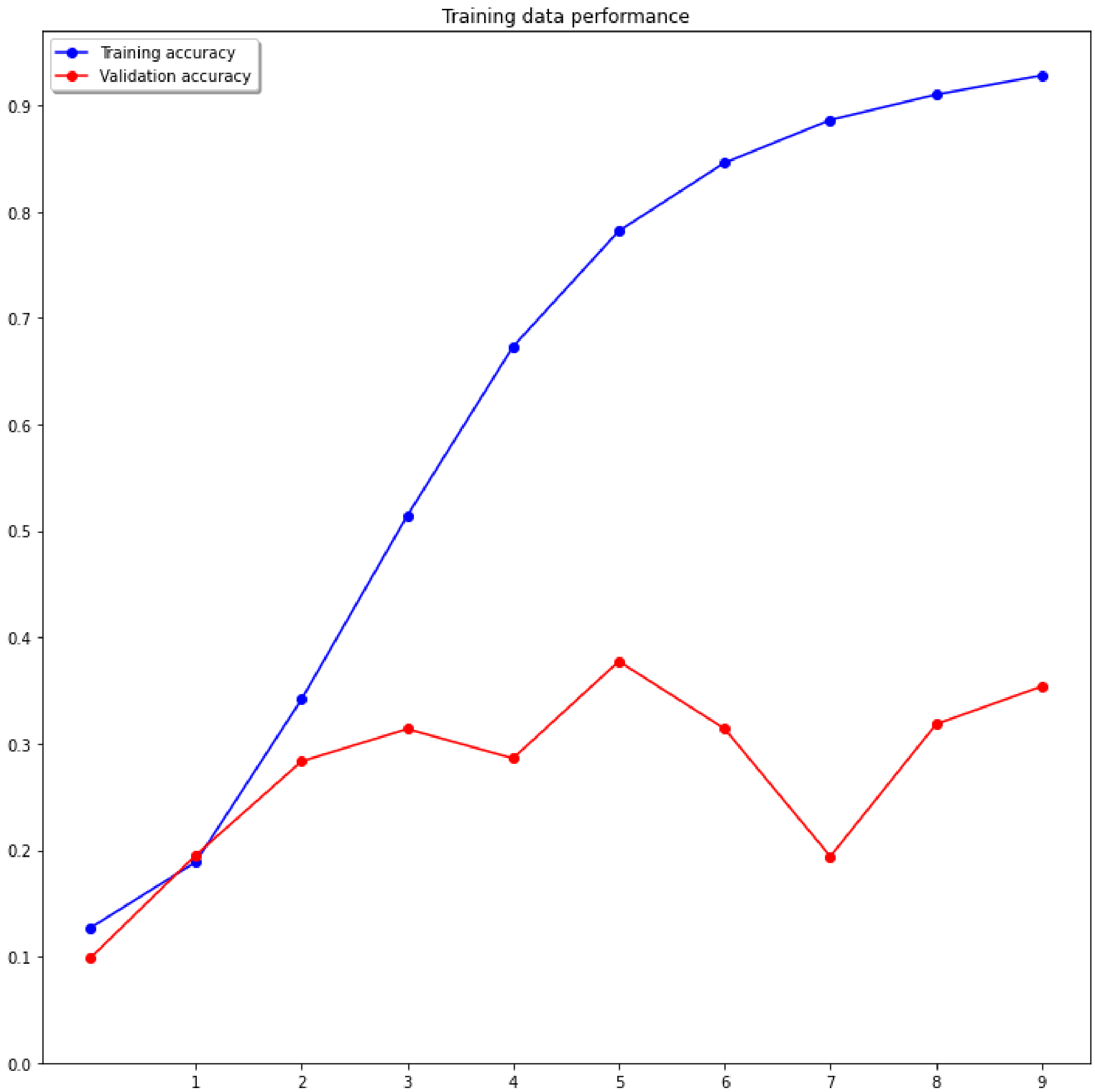
```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to
specify the frequency in number of batches seen.
Epoch 1/10
547/547 [==============================] - 204s 370ms/step - loss: 2.2992 -
accuracy: 0.1166 - val_loss: 2.2855 - val_accuracy: 0.0981

Epoch 00001: val_accuracy improved from -inf to 0.09812, saving model to
C:\Users\asebaq\distraction-detection\basic_model_aug\distraction-01-0.10.hdf5
Epoch 2/10
547/547 [==============================] - 201s 366ms/step - loss: 2.2609 -
accuracy: 0.1661 - val_loss: 2.0994 - val_accuracy: 0.1952

Epoch 00002: val_accuracy improved from 0.09812 to 0.19522, saving model to
C:\Users\asebaq\distraction-detection\basic_model_aug\distraction-02-0.20.hdf5
Epoch 3/10
547/547 [==============================] - 201s 367ms/step - loss: 1.9451 -
accuracy: 0.2966 - val_loss: 1.9285 - val_accuracy: 0.2833

Epoch 00003: val_accuracy improved from 0.19522 to 0.28332, saving model to
C:\Users\asebaq\distraction-detection\basic_model_aug\distraction-03-0.28.hdf5
Epoch 4/10
547/547 [==============================] - 199s 364ms/step - loss: 1.4647 -
accuracy: 0.4690 - val_loss: 2.3344 - val_accuracy: 0.3138
```

```
Epoch 00004: val_accuracy improved from 0.28332 to 0.31378, saving model to
C:\Users\asebaq\distraction-detection\basic_model_aug\distraction-04-0.31.hdf5
Epoch 5/10
547/547 [==============================] - 200s 365ms/step - loss: 1.0388 -
accuracy: 0.6413 - val_loss: 2.6923 - val_accuracy: 0.2862

Epoch 00005: val_accuracy did not improve from 0.31378
Epoch 6/10
547/547 [==============================] - 204s 372ms/step - loss: 0.7121 -
accuracy: 0.7621 - val_loss: 1.9957 - val_accuracy: 0.3774

Epoch 00006: val_accuracy improved from 0.31378 to 0.37735, saving model to
C:\Users\asebaq\distraction-detection\basic_model_aug\distraction-06-0.38.hdf5
Epoch 7/10
547/547 [==============================] - 202s 368ms/step - loss: 0.5080 -
accuracy: 0.8324 - val_loss: 3.0106 - val_accuracy: 0.3140

Epoch 00007: val_accuracy did not improve from 0.37735
Epoch 8/10
547/547 [==============================] - 202s 368ms/step - loss: 0.3794 -
accuracy: 0.8769 - val_loss: 3.2480 - val_accuracy: 0.1940

Epoch 00008: val_accuracy did not improve from 0.37735
Epoch 9/10
547/547 [==============================] - 201s 366ms/step - loss: 0.2983 -
accuracy: 0.9060 - val_loss: 3.1119 - val_accuracy: 0.3183

Epoch 00009: val_accuracy did not improve from 0.37735
Epoch 10/10
547/547 [==============================] - 200s 365ms/step - loss: 0.2451 -
accuracy: 0.9251 - val_loss: 2.9432 - val_accuracy: 0.3538

Epoch 00010: val_accuracy did not improve from 0.37735
```

6 Plot performance graphs

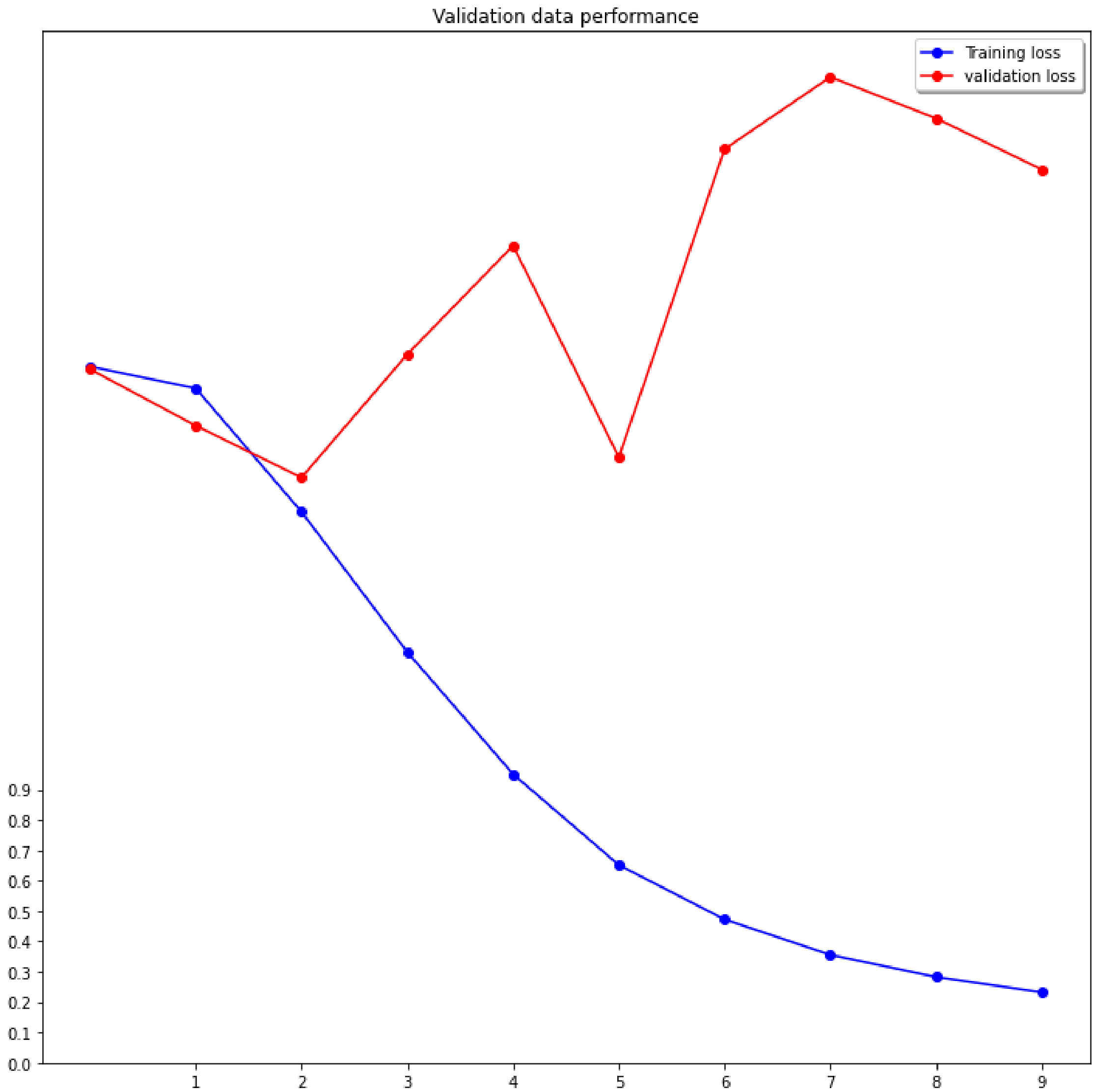
[23]:
```
# Training date
fig = plt.figure(figsize=(10, 10))
plt.plot(model_history.history['accuracy'], '-bo', label="Training accuracy")
plt.plot(model_history.history['val_accuracy'], '-ro',label="Validation␣
↪accuracy")
plt.xticks(np.arange(1, epochs, 1))
plt.yticks(np.arange(0, 1, 0.1))
legend = plt.legend(loc='best', shadow=True)
plt.title("Training data performance")
plt.tight_layout()
```

```
plt.savefig(os.path.join(MODEL_PATH, "basic_model_aug_training.png"))
plt.show()
```



[24]:
```
# Validation date
fig = plt.figure(figsize=(10, 10))
plt.plot(model_history.history['loss'], '-bo', label="Training loss")
plt.plot(model_history.history['val_loss'], '-ro', label="validation loss")
plt.xticks(np.arange(1, epochs, 1))
plt.yticks(np.arange(0, 1, 0.1))
legend = plt.legend(loc='best', shadow=True)
plt.title("Validation data performance")
plt.tight_layout()
plt.savefig(os.path.join(MODEL_PATH, "basic_model_aug_validation.png"))
```

```
plt.show()
```



7 Model Analysis

Finding the Confusion matrix, Precision, Recall and F1 score to analyse the model performance

[25]:
```
def show_confusion_matrix(confusion_matrix, class_names):
    figsize = (10,10)
    df_cm = pd.DataFrame(confusion_matrix, index=class_names,␣
 ↪columns=class_names)
    fig = plt.figure(figsize=figsize)
```

```
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")

    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,␣
 ↪ha='right')
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,␣
 ↪ha='right')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    fig.savefig(os.path.join(MODEL_PATH, "basic_model_aug_confusion_matrix.
 ↪png"))
    plt.show()
    return fig

def show_heatmap(n_labels, n_predictions, class_names):
    labels = n_labels
    predictions = n_predictions
    matrix = confusion_matrix(labels.argmax(axis=1), predictions.argmax(axis=1))
    row_sum = np.sum(matrix, axis=1)
    w, h = matrix.shape
    c_m = np.zeros((w, h))
    for i in range(h):
        c_m[i] = matrix[i] * 100 / row_sum[i]
    c = c_m.astype(dtype=np.uint8)
    heatmap = show_confusion_matrix(c, class_names)
```

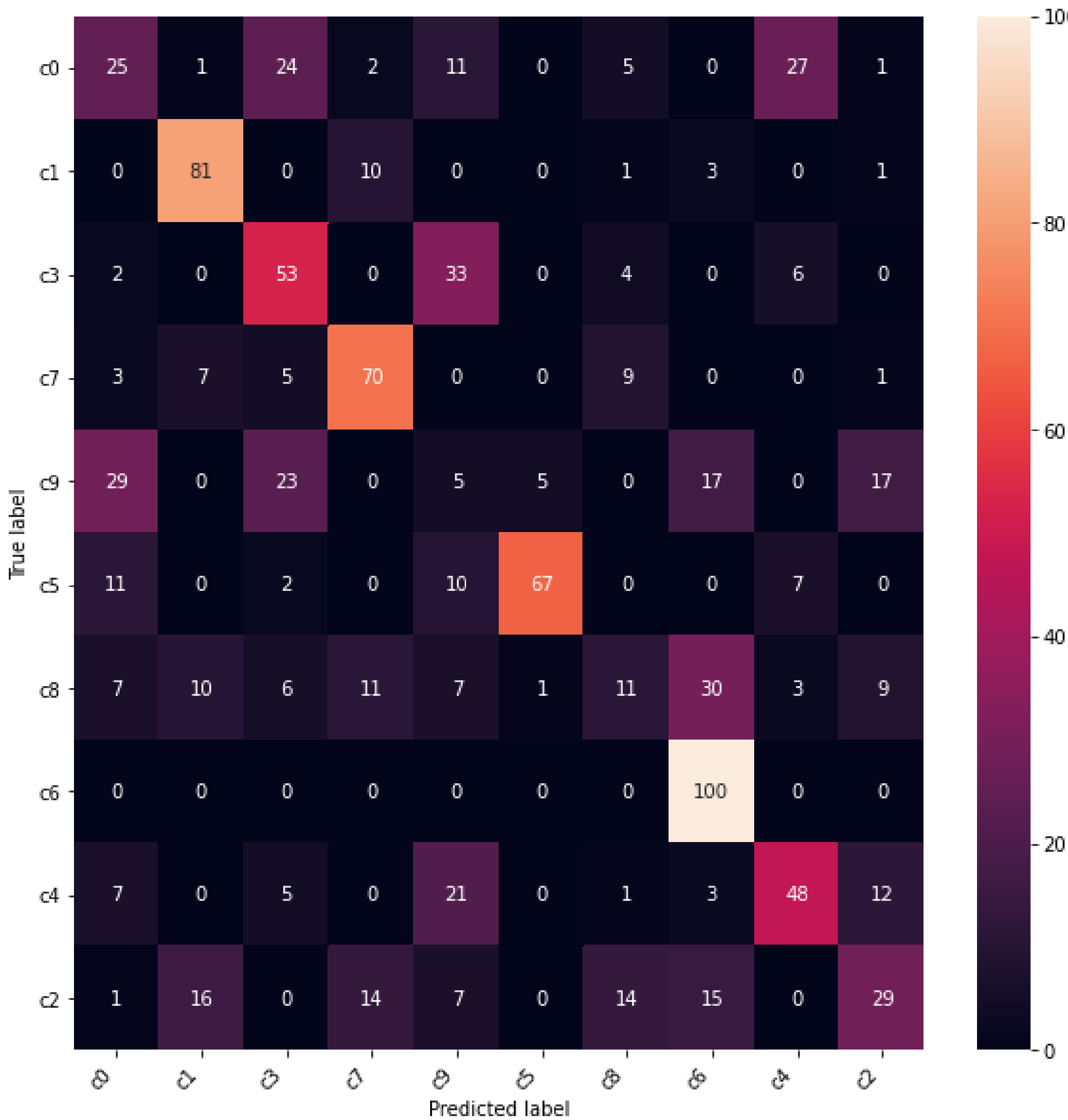
[26]:
```
class_names = list()
for name, idx in labels_id.items():
    class_names.append(name)
```

[27]:
```
# Create the validation data generator
test_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = test_datagen.flow_from_dataframe(test_df,
                                                         x_col="img",
                                                         y_col="classname",
                                                         target_size=target_size,
                                                         batch_size=1,
                                                         shuffle=False,
                                                        ␣
 ↪class_mode='categorical')
y_pred = model.predict(validation_generator)
```

```
Found 4892 validated image filenames belonging to 10 classes.
```

[28]:
```
y_test = test_df["classname"].apply(int)
y_test = np.asarray(list(y_test)).reshape((-1, 1))
y_test = to_categorical(y_test)
```

[29]:
```
show_heatmap(y_pred, y_test, class_names)
```



[30]:
```
result = model.evaluate(validation_generator)
print(f"Loss: {round(result[0], 5)}")
print(f"Accuracy: {round(result[1]*100, 3)} %")
```

```
4892/4892 [==============================] - 33s 7ms/step - loss: 2.9432 -
accuracy: 0.3538
```

Loss: 2.94318
Accuracy: 35.384 %

8 Precision, recall, and F1 score

[31]:
```
y_pred = np.argmax(y_pred, axis=1)
y_test = np.argmax(y_test, axis=1)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(accuracy, 3)}")

# precision = tp / (tp + fp)
precision = precision_score(y_test, y_pred, average='weighted')
print(f"Precision: {round(precision, 3)}")

# recall = tp / (tp + fn)
recall = recall_score(y_test, y_pred, average='weighted')
print(f"Recall: {round(recall, 3)}")

# f1 = 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"F1 score: {round(f1, 3)}")
```

```
Accuracy: 0.354
Precision: 0.501
Recall: 0.354
F1 score: 0.303
```

9 Performance on test data

[32]:
```
classes = {"c0": "safe driving",
           "c1": "texting - right",
           "c2": "talking on the phone - right",
           "c3": "texting - left",
           "c4": "talking on the phone - left",
           "c5": "operating the radio",
           "c6": "drinking",
           "c7": "reaching behind",
           "c8": "hair and makeup",
           "c9": "talking to passenger"
          }

# A function to load and resize the image
def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size= (64, 64))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
```

```
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D␣
↪tensor
    return np.expand_dims(x, axis=0)


plt.figure(figsize=(15, 15))
for i, img_path in data_test.iterrows():
    img_path = img_path.values[0]
    img = image.load_img(img_path)
    ax = plt.subplot(int(f"44{i+1}"))
    ax.margins(0.05)
    ax.imshow(img)
    img_tensor = path_to_tensor(img_path).astype('float32')/255
    label = np.argmax(model.predict(img_tensor))
    label = classes[labels_list[label]]
    ax.set_title(label)
    if i > 6:
        break
plt.savefig(os.path.join(MODEL_PATH, "basic_model_aug_test_samples.png"))
plt.show()
```

10 Use predefined models

We are also using transfer learning and fine tuning

10.1 InceptionV3

[33]:
```
MODEL_PATH = os.path.join(os.getcwd(),"inception_v3")
os.makedirs(MODEL_PATH, exist_ok=True)

# this could also be the output a different Keras model or layer
input_tensor = Input(shape=(224, 224, 3))

# create the base pre-trained model
base_model = InceptionV3(input_tensor=input_tensor, weights='imagenet',␣
↪include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(10, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False
# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])


batch_size = 32
target_size = (224, 224)
# Create the training data generator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
train_generator = train_datagen.flow_from_dataframe(train_df,
                                                    x_col="img",
                                                    y_col="classname",
                                                    target_size=target_size,
                                                    batch_size=batch_size,
                                                    class_mode="categorical")

# Create the validation data generator
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = test_datagen.flow_from_dataframe(test_df,
                                                         x_col="img",
                                                         y_col="classname",
                                                         target_size=target_size,
                                                         batch_size=batch_size,
                                                         shuffle=False,
                                                        ␣
 ↪class_mode='categorical')

steps_per_epoch = train_df.shape[0] / batch_size
validation_steps = test_df.shape[0] / batch_size
epochs = 5
# Save the model with the best accuracy during the training process
filepath = os.path.join(MODEL_PATH, "distraction-{epoch:02d}-{val_accuracy:.2f}.
 ↪hdf5")
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,␣
 ↪save_best_only=True, mode='max', period=1)
callbacks_list = [checkpoint]
# train the model on the new data for a few epochs
model_history = model.fit(train_generator,
                          steps_per_epoch=steps_per_epoch,
                          epochs=epochs,
                          validation_data=validation_generator,
                          validation_steps=validation_steps,
                          callbacks=callbacks_list)
```

```
Found 17532 validated image filenames belonging to 10 classes.
Found 4892 validated image filenames belonging to 10 classes.
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to
specify the frequency in number of batches seen.
Epoch 1/5
547/547 [==============================] - 638s 1s/step - loss: 3.0284 -
accuracy: 0.3827 - val_loss: 1.9682 - val_accuracy: 0.3761

Epoch 00001: val_accuracy improved from -inf to 0.37612, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-01-0.38.hdf5
Epoch 2/5
547/547 [==============================] - 662s 1s/step - loss: 0.7324 -
accuracy: 0.7556 - val_loss: 2.0804 - val_accuracy: 0.3698

Epoch 00002: val_accuracy did not improve from 0.37612
Epoch 3/5
547/547 [==============================] - 652s 1s/step - loss: 0.4675 -
accuracy: 0.8474 - val_loss: 1.9102 - val_accuracy: 0.4266

Epoch 00003: val_accuracy improved from 0.37612 to 0.42661, saving model to
```

```
C:\Users\asebaq\distraction-detection\inception_v3\distraction-03-0.43.hdf5
Epoch 4/5
547/547 [==============================] - 692s 1s/step - loss: 0.3591 -
accuracy: 0.8840 - val_loss: 2.4716 - val_accuracy: 0.4109

Epoch 00004: val_accuracy did not improve from 0.42661
Epoch 5/5
547/547 [==============================] - 742s 1s/step - loss: 0.3064 -
accuracy: 0.9019 - val_loss: 1.6987 - val_accuracy: 0.5125

Epoch 00005: val_accuracy improved from 0.42661 to 0.51247, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-05-0.51.hdf5
```

[34]:

```python
# at this point, the top layers are well trained and we can start fine-tuning
# convolutional layers from inception V3. We will freeze the bottom N layers
# and train the remaining top layers.

# let's visualize layer names and layer indices to see how many layers
# we should freeze:
# for i, layer in enumerate(base_model.layers):
#     print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model.layers[:249]:
    layer.trainable = False

for layer in model.layers[249:]:
    layer.trainable = True

# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])

# we train our model again (this time fine-tuning the top 2 inception blocks
# alongside the top Dense layers
epochs = 10
model_history = model.fit(train_generator,
                          steps_per_epoch=steps_per_epoch,
                          epochs=epochs,
                          validation_data=validation_generator,
                          validation_steps=validation_steps,
                          callbacks=callbacks_list)
```

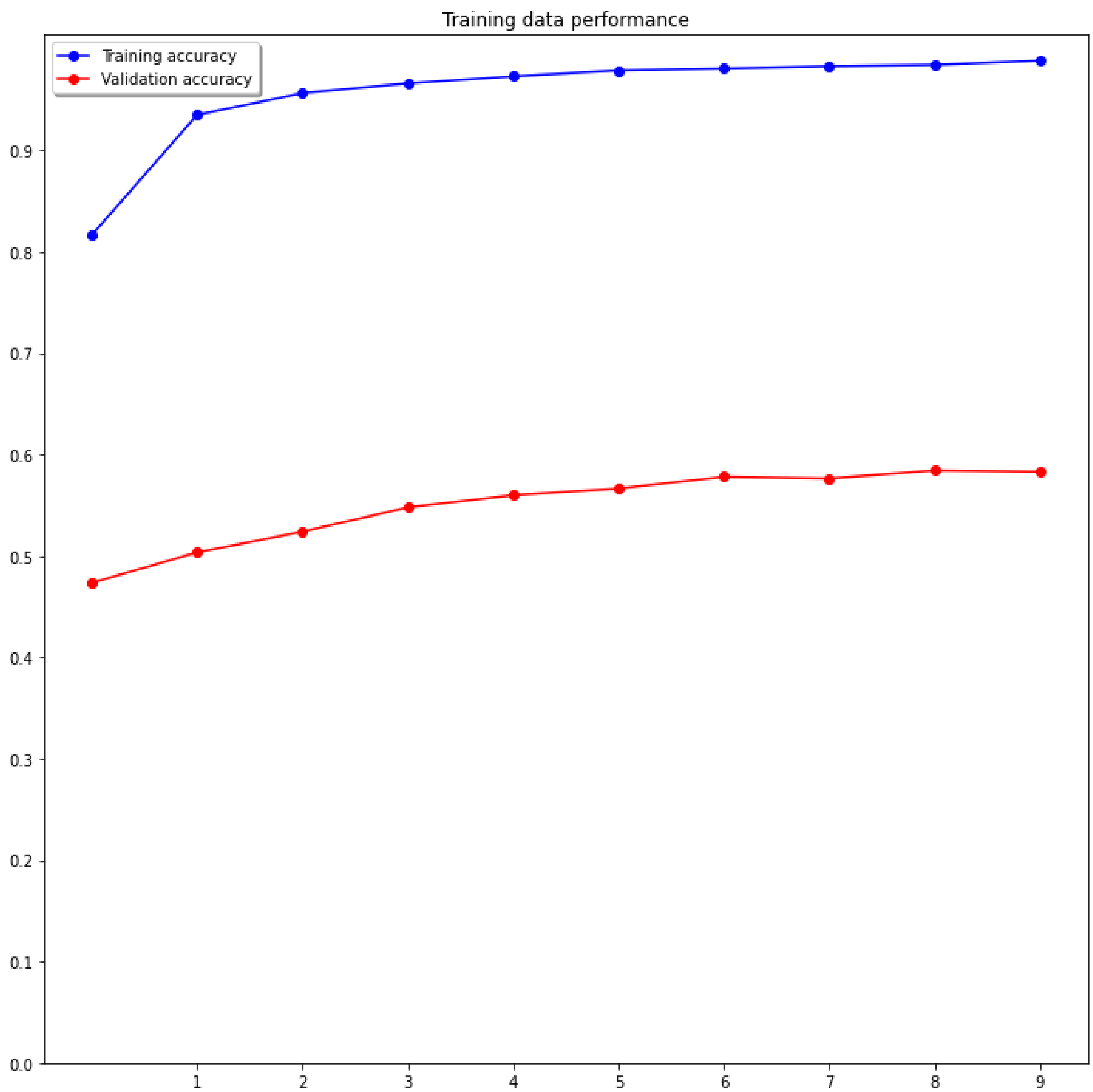
```
Epoch 1/10
547/547 [==============================] - 905s 2s/step - loss: 0.9126 -
accuracy: 0.7086 - val_loss: 1.5380 - val_accuracy: 0.4734
```

```
Epoch 00001: val_accuracy did not improve from 0.51247
Epoch 2/10
547/547 [==============================] - 799s 1s/step - loss: 0.2832 -
accuracy: 0.9274 - val_loss: 1.4636 - val_accuracy: 0.5035

Epoch 00002: val_accuracy did not improve from 0.51247
Epoch 3/10
547/547 [==============================] - 829s 2s/step - loss: 0.1801 -
accuracy: 0.9568 - val_loss: 1.4293 - val_accuracy: 0.5239

Epoch 00003: val_accuracy improved from 0.51247 to 0.52392, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-03-0.52.hdf5
Epoch 4/10
547/547 [==============================] - 860s 2s/step - loss: 0.1375 -
accuracy: 0.9632 - val_loss: 1.3612 - val_accuracy: 0.5478

Epoch 00004: val_accuracy improved from 0.52392 to 0.54783, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-04-0.55.hdf5
Epoch 5/10
547/547 [==============================] - 847s 2s/step - loss: 0.1120 -
accuracy: 0.9703 - val_loss: 1.3236 - val_accuracy: 0.5599

Epoch 00005: val_accuracy improved from 0.54783 to 0.55989, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-05-0.56.hdf5
Epoch 6/10
547/547 [==============================] - 847s 2s/step - loss: 0.0880 -
accuracy: 0.9799 - val_loss: 1.3192 - val_accuracy: 0.5664

Epoch 00006: val_accuracy improved from 0.55989 to 0.56643, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-06-0.57.hdf5
Epoch 7/10
547/547 [==============================] - 875s 2s/step - loss: 0.0770 -
accuracy: 0.9810 - val_loss: 1.3019 - val_accuracy: 0.5781

Epoch 00007: val_accuracy improved from 0.56643 to 0.57809, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-07-0.58.hdf5
Epoch 8/10
547/547 [==============================] - 847s 2s/step - loss: 0.0690 -
accuracy: 0.9826 - val_loss: 1.3141 - val_accuracy: 0.5765

Epoch 00008: val_accuracy did not improve from 0.57809
Epoch 9/10
547/547 [==============================] - 875s 2s/step - loss: 0.0640 -
accuracy: 0.9853 - val_loss: 1.2946 - val_accuracy: 0.5842

Epoch 00009: val_accuracy improved from 0.57809 to 0.58422, saving model to
C:\Users\asebaq\distraction-detection\inception_v3\distraction-09-0.58.hdf5
```

```
Epoch 10/10
547/547 [==============================] - 840s 2s/step - loss: 0.0530 -
accuracy: 0.9869 - val_loss: 1.3143 - val_accuracy: 0.5830

Epoch 00010: val_accuracy did not improve from 0.58422
```

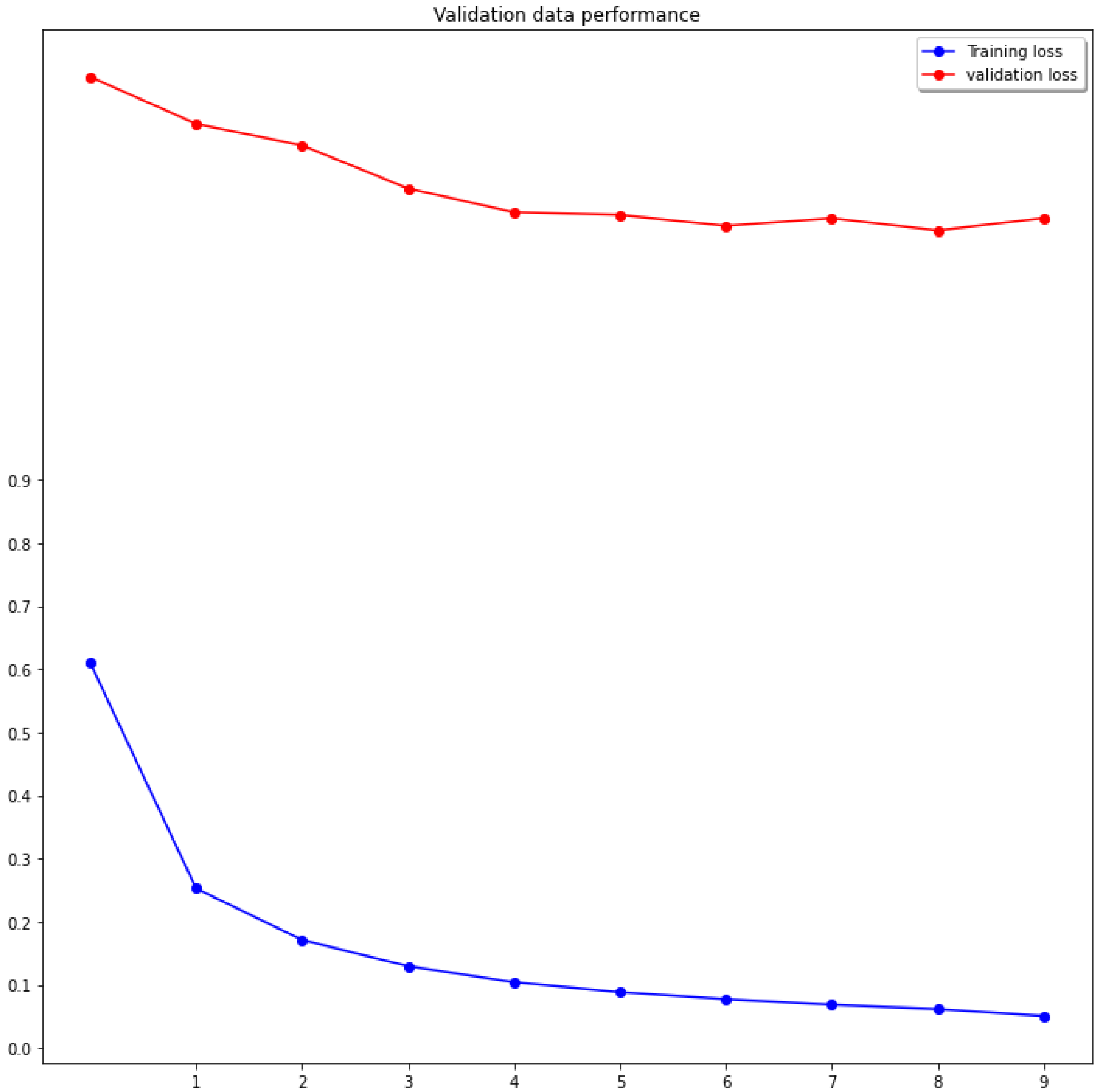
11 Plot performance graphs

[35]:
```
# Training date
fig = plt.figure(figsize=(10, 10))
plt.plot(model_history.history['accuracy'], '-bo', label="Training accuracy")
plt.plot(model_history.history['val_accuracy'], '-ro',label="Validation␣
↪accuracy")
plt.xticks(np.arange(1, epochs, 1))
plt.yticks(np.arange(0, 1, 0.1))
legend = plt.legend(loc='best', shadow=True)
plt.title("Training data performance")
plt.tight_layout()
plt.savefig(os.path.join(MODEL_PATH, "inceptionv3_model_training.png"))
plt.show()
```



[36]:

```
# Validation date
fig = plt.figure(figsize=(10, 10))
plt.plot(model_history.history['loss'], '-bo', label="Training loss")
plt.plot(model_history.history['val_loss'], '-ro', label="validation loss")
plt.xticks(np.arange(1, epochs, 1))
plt.yticks(np.arange(0, 1, 0.1))
legend = plt.legend(loc='best', shadow=True)
plt.title("Validation data performance")
plt.tight_layout()
plt.savefig(os.path.join(MODEL_PATH, "inceptionv3_model_validation.png"))
plt.show()
```



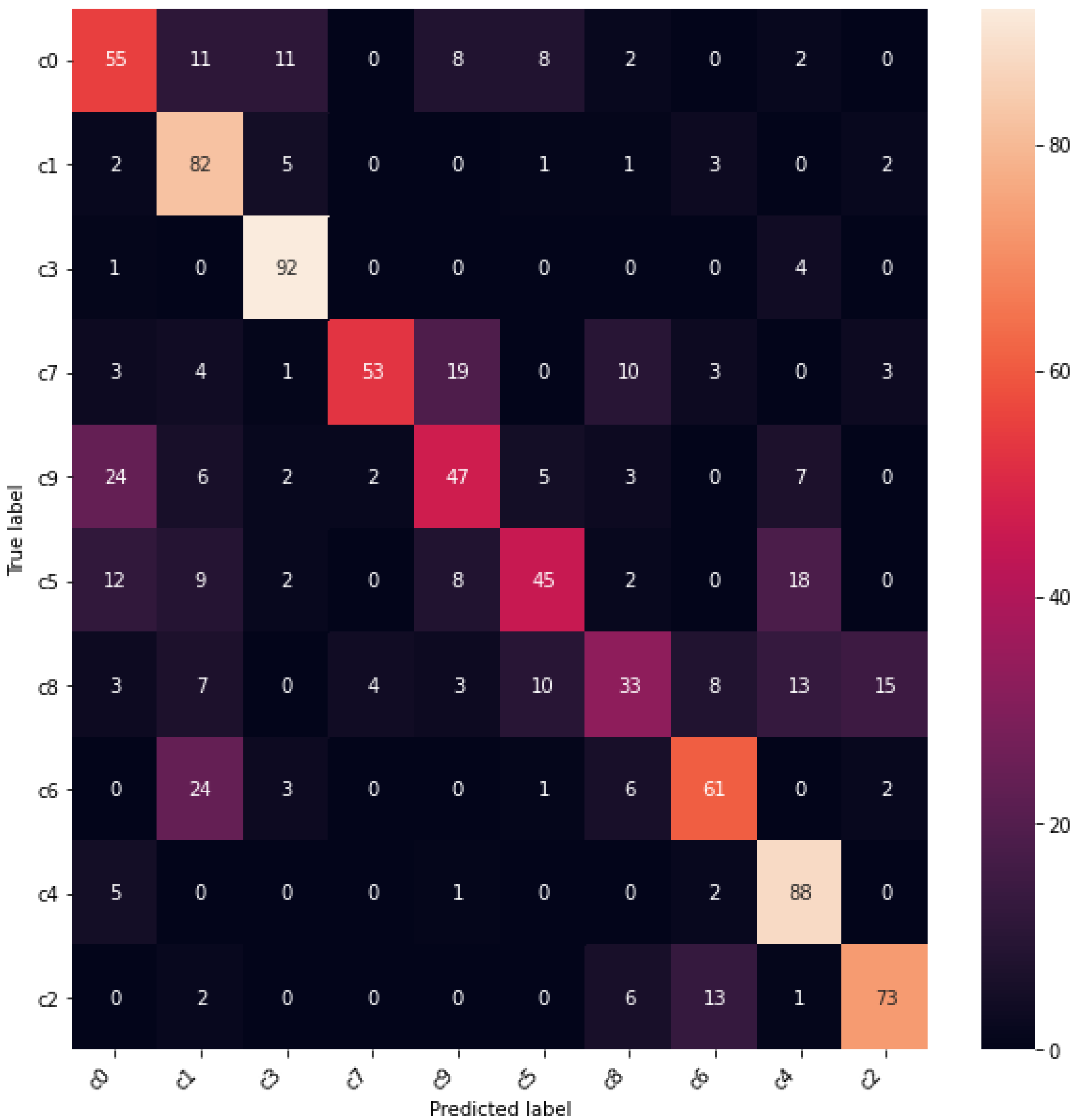
12 InceptionV3 model analysis

[37]:
```
def show_confusion_matrix(confusion_matrix, class_names):
    figsize = (10,10)
    df_cm = pd.DataFrame(confusion_matrix, index=class_names,␣
 ↪columns=class_names)
    fig = plt.figure(figsize=figsize)

    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
```

```
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0,␣
 ↪ha='right')
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45,␣
 ↪ha='right')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    fig.savefig(os.path.join(MODEL_PATH, "inceptionv3_aug_confusion_matrix.
 ↪png"))
    plt.show()
    return fig

def show_heatmap(n_labels, n_predictions, class_names):
    labels = n_labels
    predictions = n_predictions
    matrix = confusion_matrix(labels.argmax(axis=1), predictions.argmax(axis=1))
    row_sum = np.sum(matrix, axis=1)
    w, h = matrix.shape
    c_m = np.zeros((w, h))
    for i in range(h):
        c_m[i] = matrix[i] * 100 / row_sum[i]
    c = c_m.astype(dtype=np.uint8)
    heatmap = show_confusion_matrix(c, class_names)

# Create the validation data generator
test_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = test_datagen.flow_from_dataframe(test_df,
                                                         x_col="img",
                                                         y_col="classname",
                                                         target_size=target_size,
                                                         batch_size=1,
                                                         shuffle=False,
                                                        ␣
 ↪class_mode='categorical')
y_pred = model.predict(validation_generator)
y_test = test_df["classname"].apply(int)
y_test = np.asarray(list(y_test)).reshape((-1, 1))
y_test = to_categorical(y_test)
show_heatmap(y_pred, y_test, class_names)
result = model.evaluate(validation_generator)
print(f"Loss: {round(result[0], 5)}")
print(f"Accuracy: {round(result[1]*100, 3)} %")

y_pred = np.argmax(y_pred, axis=1)
y_test = np.argmax(y_test, axis=1)
```

```python
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {round(accuracy, 3)}")

# precision = tp / (tp + fp)
precision = precision_score(y_test, y_pred, average='weighted')
print(f"Precision: {round(precision, 3)}")

# recall = tp / (tp + fn)
recall = recall_score(y_test, y_pred, average='weighted')
print(f"Recall: {round(recall, 3)}")

# f1 = 2 tp / (2 tp + fp + fn)
f1 = f1_score(y_test, y_pred, average='weighted')
print(f"F1 score: {round(f1, 3)}")
```

```
Found 4892 validated image filenames belonging to 10 classes.
```




```
4892/4892 [==============================] - 229s 47ms/step - loss: 1.3144 -
accuracy: 0.5830s - loss: 1.3148 - accu
Loss: 1.31435
Accuracy: 58.299 %
Accuracy: 0.583
Precision: 0.644
Recall: 0.583
F1 score: 0.58
```

13 Performance on test data

[38]:
```python
classes = {"c0": "safe driving",
           "c1": "texting - right",
           "c2": "talking on the phone - right",
           "c3": "texting - left",
           "c4": "talking on the phone - left",
           "c5": "operating the radio",
           "c6": "drinking",
           "c7": "reaching behind",
           "c8": "hair and makeup",
           "c9": "talking to passenger"
          }

# A function to load and resize the image
def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size= (224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D␣
↪tensor
    return np.expand_dims(x, axis=0)


plt.figure(figsize=(15, 15))
for i, img_path in data_test.iterrows():
    img_path = img_path.values[0]
    img = image.load_img(img_path)
    ax = plt.subplot(int(f"44{i+1}"))
    ax.margins(0.05)
    ax.imshow(img)
    img_tensor = path_to_tensor(img_path).astype('float32')/255
    label = np.argmax(model.predict(img_tensor))
    label = classes[labels_list[label]]
    ax.set_title(label)
    if i > 6:
        break
plt.savefig(os.path.join(MODEL_PATH, "inception_model_test_samples.png"))
plt.show()
```

13.0.1 We can also try to use ResNet50V2, MobileNetV2, or VGG16.

13.0.2 And may be do an ensembled model which will combine all of these.

[ ]:

```
```

[ ]:

```
```