In [195]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Conv2D, Activation, MaxPool2D, BatchNormalization, Flatten, Dense, Dropout
import pathlib
import scipy.io
import os, os.path, shutil
import pandas as pd
from PIL import Image
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [208]:
```
def create_classes_folder(path_):
    images = [f for f in os.listdir(path_) if os.path.isfile(os.path.join(path_, f))]
    for image in images:
        folder_name = df[df.image == image]['class'].values[0]
        new_path = os.path.join(folder_path, folder_name)
        if not os.path.exists(new_path):
            os.makedirs(new_path)

        old_image_path = os.path.join(folder_path, image)
        new_image_path = os.path.join(new_path, image)
        shutil.move(old_image_path, new_image_path)
```

In [203]:
```
def make_model(input_shape, num_classes, model_type_, aug_version_):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    if aug_version_ == 1:
        x = data_augmentation(inputs)
    elif aug_version_ == 2:
        x = aug(inputs)

    # Entry block
    x = layers.Rescaling(1.0 / 255)(x)
    if model_type_ == 1:
        x = layers.Conv2D(16, 3, padding='same', activation='relu')(x)
        x = layers.MaxPooling2D()(x)
        x = layers.Conv2D(32, 3, padding='same', activation='relu')(x)
        x = layers.MaxPooling2D()(x)
        x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
        x = layers.MaxPooling2D()(x)
        x = layers.Dropout(0.2)(x)
        x = layers.Flatten()(x)
        x = layers.Dense(128, activation='relu')(x)
    elif model_type_ == 2:
        x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation("relu")(x)

        x = layers.Conv2D(64, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation("relu")(x)

        previous_block_activation = x  # Set aside residual

        for size in [128, 256, 512, 728]:
            x = layers.Activation("relu")(x)
            x = layers.SeparableConv2D(size, 3, padding="same")(x)
            x = layers.BatchNormalization()(x)

            x = layers.Activation("relu")(x)
            x = layers.SeparableConv2D(size, 3, padding="same")(x)
            x = layers.BatchNormalization()(x)

            x = layers.MaxPooling2D(3, strides=2, padding="same")(x)
```

```
            # Project residual
            residual = layers.Conv2D(size, 1, strides=2, padding="same")(
                previous_block_activation
            )
            x = layers.add([x, residual])
            previous_block_activation = x

        x = layers.SeparableConv2D(1024, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation("relu")(x)

        x = layers.GlobalAveragePooling2D()(x)
        x = layers.Dropout(0.5)(x)


    activation = "softmax"
    units = num_classes
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)
```

In [206]:
```
def plot_metrics(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
```

In [209]:
```
def flatten_list(list_):
    flat_list = []
    for sublist in list_:
        for item in sublist:
            flat_list.append(item)
    return flat_list

def make_confusion_mtrx(model_, val_ds, save_path):
    y_pred = []
    y_true = []
    for x, y in val_ds.take(-1):
        y_pred.append(np.argmax(model_.predict(x), axis = 1))
        y_true.append(y)

    y_pred_fl = flatten_list(y_pred)
    y_true_fl = flatten_list(y_true)
    conf_mx = confusion_matrix(y_pred_fl, y_true_fl)
    df_cm = pd.DataFrame(conf_mx , index = [i for i in range(0,196)],
                 columns = [i for i in range(0,196)])
    cmap = sns.diverging_palette(230, 20, as_cmap=True)
    plt.figure(figsize = (100,100))
    cm_fig = sns.heatmap(df_cm, cmap=cmap, annot=True)
    fig = cm_fig.get_figure()
    fig.savefig(save_path)
    return cm_fig
```

In [210]:
```
def make_a_prediction(model_, img_path):
    img = keras.preprocessing.image.load_img(
    img_path, target_size=(256,256))
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)  # Create batch axis

    predictions = model_.predict(img_array)
    score = predictions[0]
    image = Image.open(img_path)
    display(image)
    return (f'The car on the image is most likely - {train_ds.class_names[np.argmax(score)]}')
```

In [5]:
```
tf.__version__
```

Out[5]:
```
'2.6.0'
```

In [2]:
```
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)

for device in physical_devices:
    tf.config.experimental.set_memory_growth(device, True)
```

In [49]:
```
# uncomment for downloading data
# data_dir = tf.keras.utils.get_file(
#     "stanford_cars",
#     "http://ai.stanford.edu/~jkrause/car196/car_ims.tgz",
#     untar=True, extract= True)
# data_dir = pathlib.Path(data_dir)
```

In [5]:
```
#data_dir = pathlib.Path(data_dir)
data_dir = 'C:/Users/kaara/.keras/datasets/car_ims'
image_count = len(list(data_dir.glob('*/*.jpg')))
mat = scipy.io.loadmat('C:/Users/kaara/.keras/datasets/cars_annos.mat')
print(image_count)
```

```
16103
```

In [7]:
```
data = []
for i in range (0,mat['annotations'].shape[1]):
    obj = mat['annotations'][0][i]
    im_path = obj[0][0].split('/')[1]
    class_name = mat['class_names'][0][obj[5][0][0]-1][0]
    data.append([im_path,class_name])
df = pd.DataFrame(data, columns = ['image','class'])
```

In [69]:
```
# Uncomment when new data has been downloaded
# This step is needed because in archive all data store in one folder,
# to use tf.image_dataset_from_directory we need folder for each class

#create_classes_folder('C:/Users/kaara/.keras/datasets/car_ims')
```

In [70]:
```
image_size = (256,256)
batch_size = 16

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 16185 files belonging to 196 classes.
Using 12948 files for training.
Found 16185 files belonging to 196 classes.
Using 3237 files for validation.
```

In [196]:

```
plt.figure(figsize=(16, 16))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(train_ds.class_names[(int(labels[i]))])
        plt.axis("off")
```

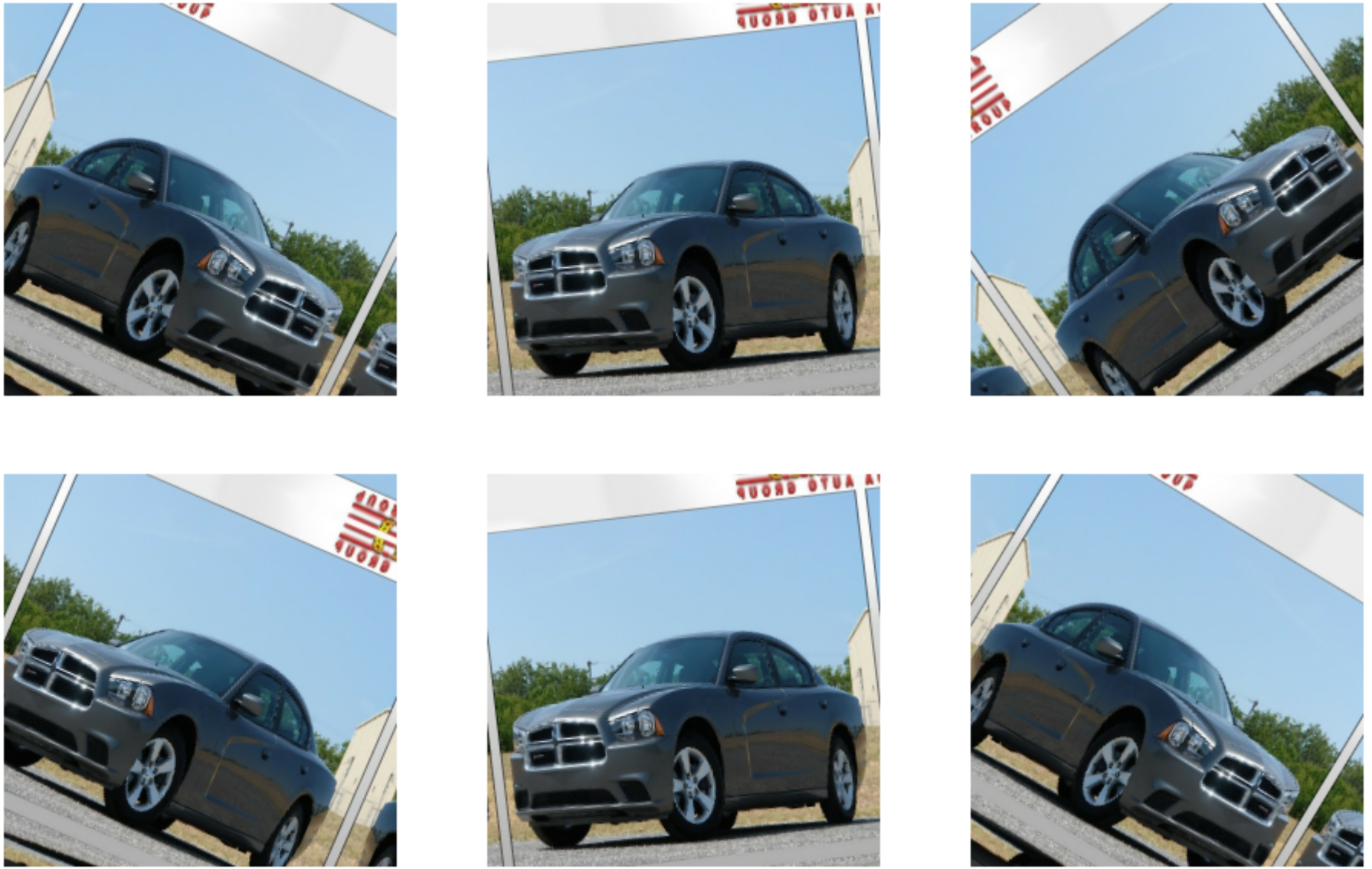
Chrysler Town and Country Minivan 2012

Tesla Model S Sedan 2012

Bentley Arnage Sedan 2009

In [198]:
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
    ]
)
```

In [199]:
```
plt.figure(figsize=(16, 16))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```





In [20]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [204]:
```
model = make_model(input_shape=image_size + (3,), num_classes=196, model_type_= 1, aug_version_= 1)
model.summary()
```

```
max_pooling2d_4 (MaxPooling2 (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 128, 128, 32)      4640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 65536)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8388736
_________________________________________________________________
dense_2 (Dense)              (None, 196)               25284
=================================================================
Total params: 8,437,604
Trainable params: 8,437,604
```

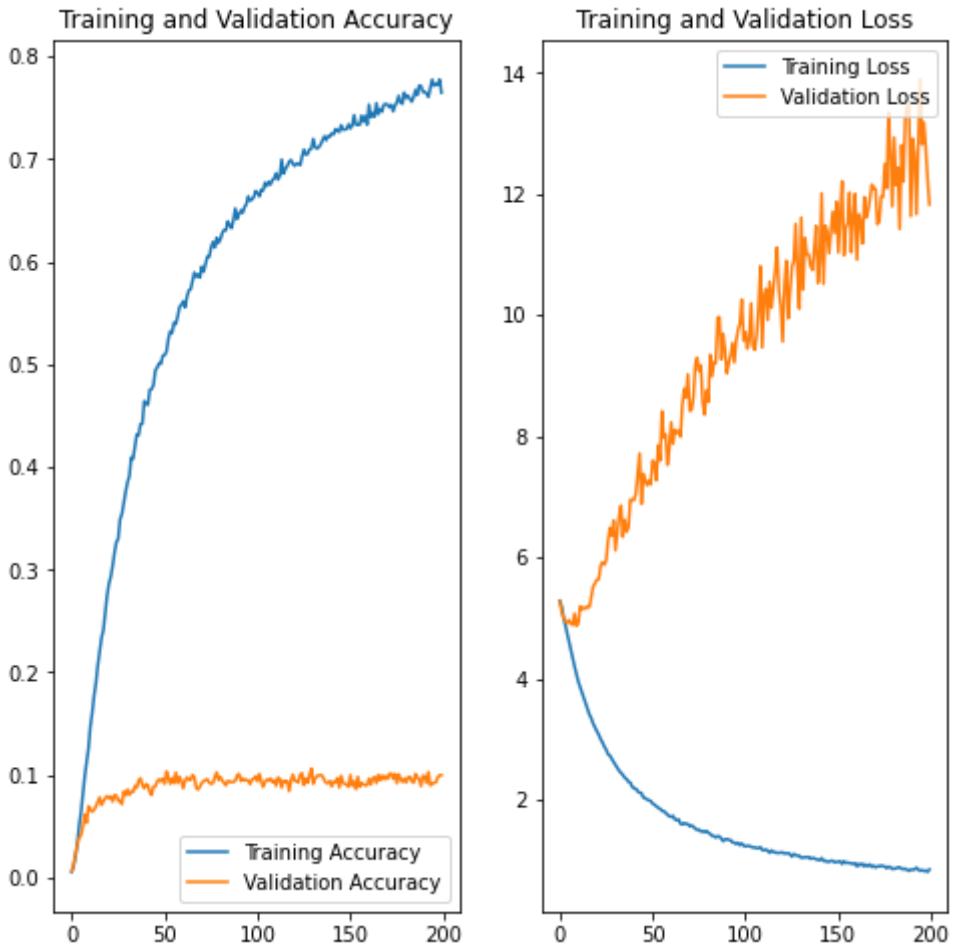
In [2]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [55]:
```
epochs = 200


model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"],
)
history_smp = model.fit(
    train_ds, epochs=epochs, validation_data=val_ds,
)
```

```
810/810 [==============================] - 19s 24ms/step - loss: 1.0913 - accuracy: 0.7049 - val_loss: 11.5021 - val_
accuracy: 0.0924
Epoch 129/200
810/810 [==============================] - 19s 23ms/step - loss: 1.0724 - accuracy: 0.7093 - val_loss: 10.7139 - val_
accuracy: 0.1001
Epoch 130/200
810/810 [==============================] - 19s 23ms/step - loss: 1.0761 - accuracy: 0.7100 - val_loss: 10.1031 - val_
accuracy: 0.1063
Epoch 131/200
810/810 [==============================] - 19s 23ms/step - loss: 1.0411 - accuracy: 0.7195 - val_loss: 11.6080 - val_
accuracy: 0.0914
Epoch 132/200
810/810 [==============================] - 19s 23ms/step - loss: 1.0470 - accuracy: 0.7117 - val_loss: 10.4184 - val_
accuracy: 0.0964
Epoch 133/200
810/810 [==============================] - 19s 23ms/step - loss: 1.0541 - accuracy: 0.7101 - val_loss: 11.2696 - val_
accuracy: 0.0992
Epoch 134/200
810/810 [==============================] - 19s 23ms/step - loss: 1.0474 - accuracy: 0.7115 - val_loss: 11.0040 - val_
accuracy: 0.0995
```

In [56]:
```
plot_metrics(history_smp)
```



That was just a simple CNN, which, actualy didn't succed on the validation data. So let's build more complex one.

In [57]:
```
model_res = make_model(input_shape=image_size + (3,), num_classes=196, model_type_= 2, aug_version_= 1)
model_res.summary()
```

```
max_pooling2d_42 (MaxPooling2D) (None, 32, 32, 256)  0           batch_normalization_60[0][0]
conv2d_54 (Conv2D)              (None, 32, 32, 256)  33024       add_20[0][0]
add_21 (Add)                    (None, 32, 32, 256)  0           max_pooling2d_42[0][0]
                                                                 conv2d_54[0][0]
activation_61 (Activation)      (None, 32, 32, 256)  0           add_21[0][0]
separable_conv2d_49 (SeparableC (None, 32, 32, 512)  133888      activation_61[0][0]
batch_normalization_61 (BatchNo (None, 32, 32, 512)  2048        separable_conv2d_49[0][0]
activation_62 (Activation)      (None, 32, 32, 512)  0           batch_normalization_61[0][0]
separable_conv2d_50 (SeparableC (None, 32, 32, 512)  267264      activation_62[0][0]
batch_normalization_62 (BatchNo (None, 32, 32, 512)  2048        separable_conv2d_50[0][0]
```

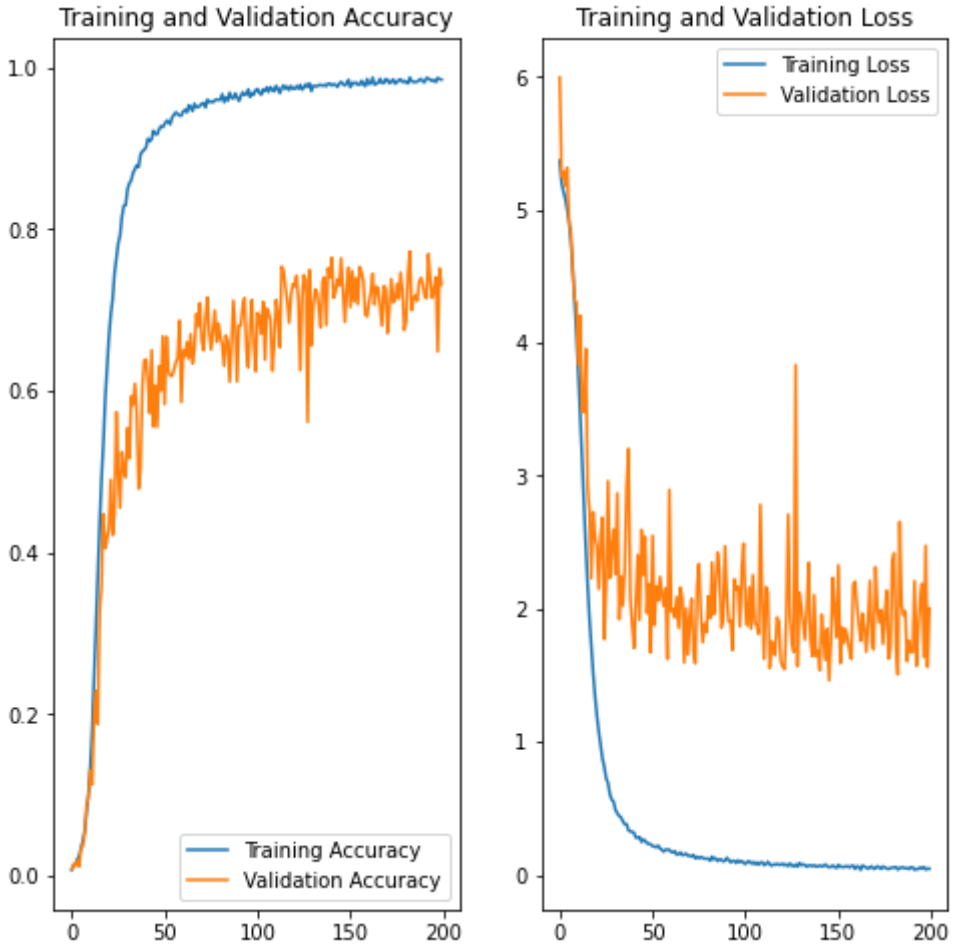
```
Initialy, I've ran it on 200 epochs, and it has taken ~10 hours to train. In case, if something goes wrong with cells, results can be found in images, which I've put into the archive.
```

Results: loss: 0.0468 - accuracy: 0.9852 - val_loss: 2.0041 - val_accuracy: 0.7322

In [59]:
```
epochs = 200

model_res.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"],
)
history = model_res.fit(
    train_ds, epochs=epochs, validation_data=val_ds,
)
```

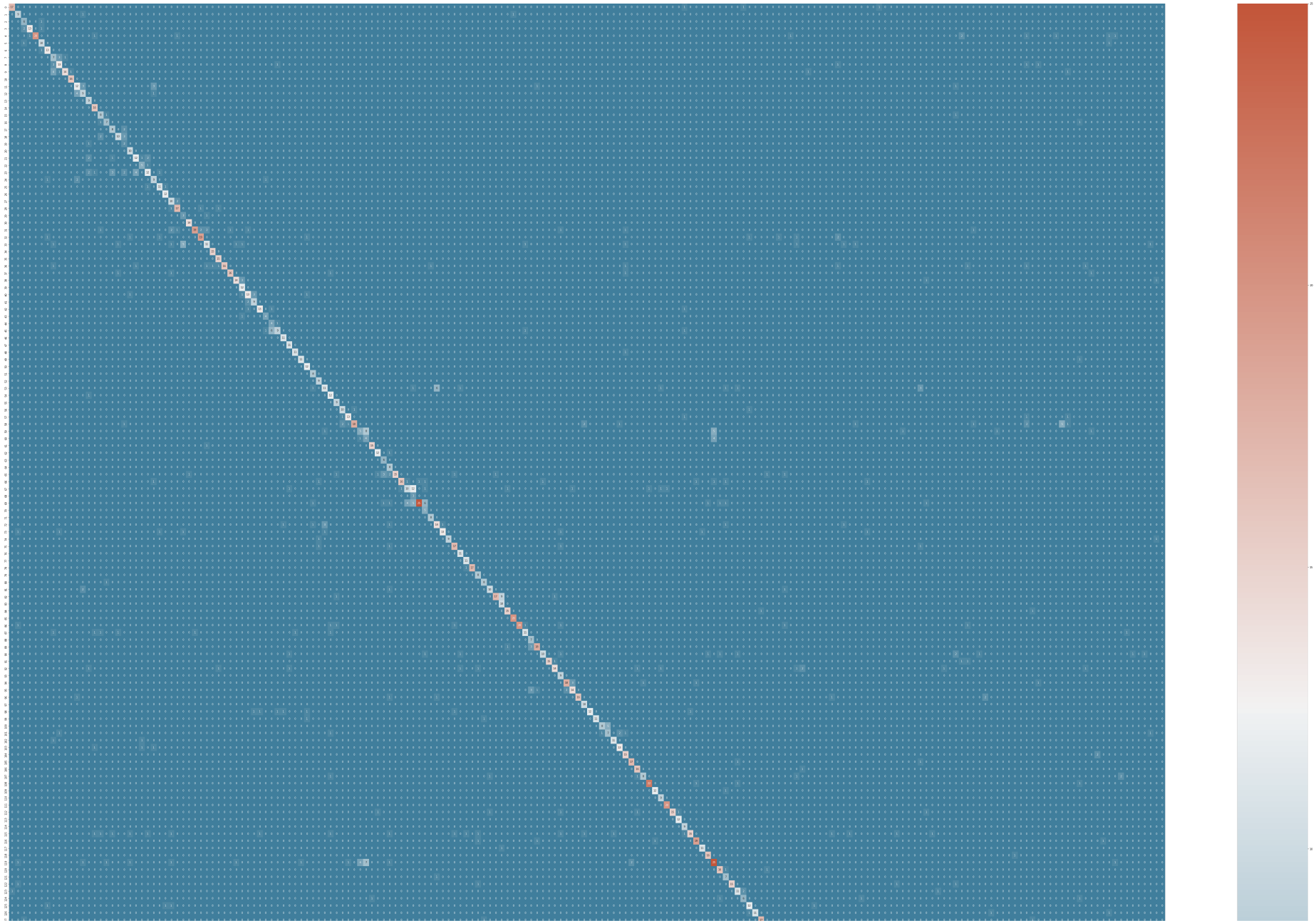
```
_accuracy: 0.7646
Epoch 142/200
810/810 [==============================] - 165s 204ms/step - loss: 0.0643 - accuracy: 0.9795 - val_loss: 1.9566 - val
_accuracy: 0.7152
Epoch 143/200
810/810 [==============================] - 164s 203ms/step - loss: 0.0675 - accuracy: 0.9788 - val_loss: 1.7729 - val
_accuracy: 0.7223
Epoch 144/200
810/810 [==============================] - 165s 204ms/step - loss: 0.0788 - accuracy: 0.9761 - val_loss: 1.6163 - val
_accuracy: 0.7377
Epoch 145/200
810/810 [==============================] - 164s 203ms/step - loss: 0.0649 - accuracy: 0.9801 - val_loss: 1.8503 - val
_accuracy: 0.7303
Epoch 146/200
810/810 [==============================] - 163s 202ms/step - loss: 0.0577 - accuracy: 0.9824 - val_loss: 1.4636 - val
_accuracy: 0.7640
Epoch 147/200
810/810 [==============================] - 167s 206ms/step - loss: 0.0613 - accuracy: 0.9798 - val_loss: 1.7593 - val
_accuracy: 0.7257
Epoch 148/200
```

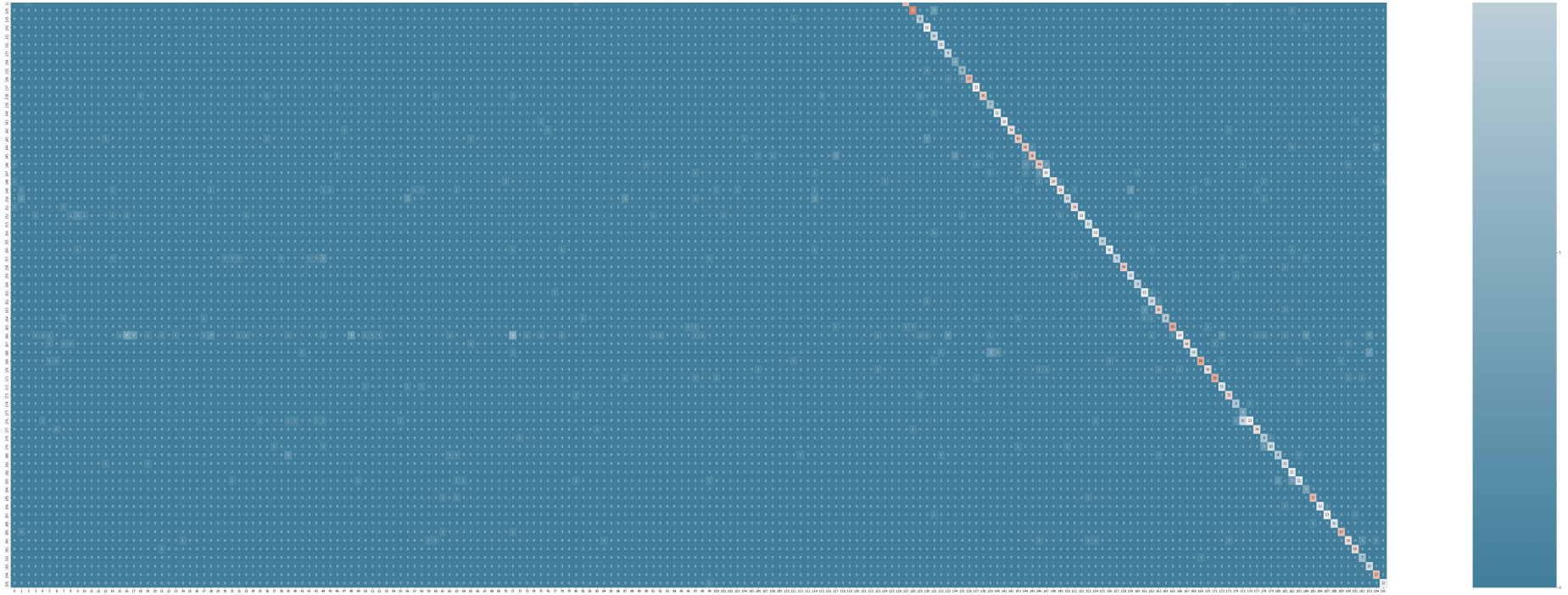
In [60]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss

In [81]:
```
make_confusion_mtrx(model_ = model_res, validation_df = val_ds, save_path = 'confusion_matrix.jpg')
```

Out[81]: <AxesSubplot:>





Hint: click on the matrix, it is much bigger, than Jupier can display in the cell

In [82]:
```
fig = cm_fig.get_figure()
fig.savefig('confusion_matrix.png')
```

I've notices that the last 50 epochs hasn't make a great influence on overall accuracy, so I've retraindes the model on 150 epochs and add sligthly more complex data augmentation

In [10]:
```
aug = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.5,0.3)
    ]
)
```

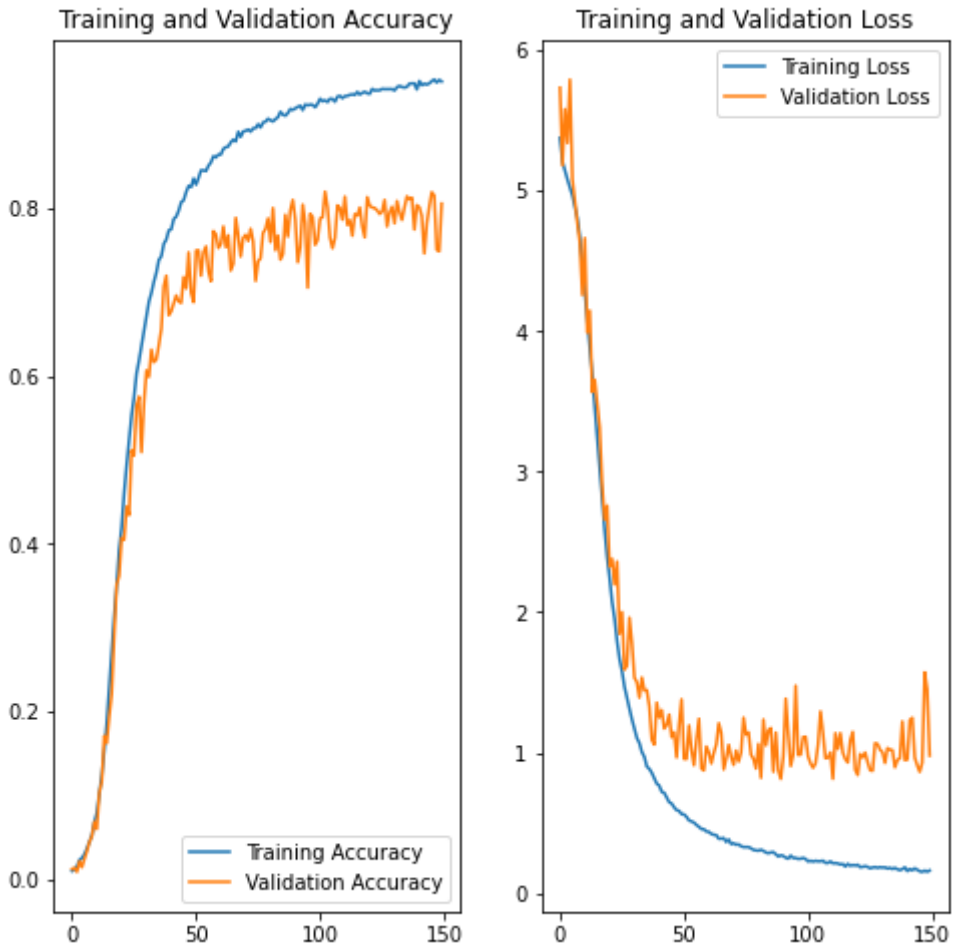
In [205]:
```
model_res_new = make_model(input_shape=image_size + (3,), num_classes=196, model_type_= 2, aug_version_= 2)
model_res_new.summary()
```

conv2d_13 (Conv2D)	(None, 16, 16, 512)	131584	add_5[0][0]
add_6 (Add)	(None, 16, 16, 512)	0	max_pooling2d_9[0][0] conv2d_13[0][0]
activation_19 (Activation)	(None, 16, 16, 512)	0	add_6[0][0]
separable_conv2d_15 (SeparableC	(None, 16, 16, 728)	378072	activation_19[0][0]
batch_normalization_19 (BatchNo	(None, 16, 16, 728)	2912	separable_conv2d_15[0][0]
activation_20 (Activation)	(None, 16, 16, 728)	0	batch_normalization_19[0][0]
separable_conv2d_16 (SeparableC	(None, 16, 16, 728)	537264	activation_20[0][0]
batch_normalization_20 (BatchNo	(None, 16, 16, 728)	2912	separable_conv2d_16[0][0]
max_pooling2d_10 (MaxPooling2D)	(None, 8, 8, 728)	0	batch_normalization_20[0][0]

In [13]:
```
epochs = 150

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]

model_res_new.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"],
)
history_new_res = model_res_new.fit(
    train_ds, epochs=epochs,callbacks=callbacks, validation_data=val_ds,
)
```

```
_accuracy: 0.7748
Epoch 80/150
810/810 [==============================] - 169s 209ms/step - loss: 0.3048 - accuracy: 0.9061 - val_loss: 0.8870 - val
_accuracy: 0.7875
Epoch 81/150
810/810 [==============================] - 175s 216ms/step - loss: 0.3023 - accuracy: 0.9043 - val_loss: 1.0626 - val
_accuracy: 0.7593
Epoch 82/150
810/810 [==============================] - 170s 210ms/step - loss: 0.3089 - accuracy: 0.9030 - val_loss: 0.8206 - val
_accuracy: 0.7998
Epoch 83/150
810/810 [==============================] - 167s 206ms/step - loss: 0.3007 - accuracy: 0.9057 - val_loss: 1.2359 - val
_accuracy: 0.7495
Epoch 84/150
810/810 [==============================] - 167s 206ms/step - loss: 0.2895 - accuracy: 0.9079 - val_loss: 1.0517 - val
_accuracy: 0.7671
Epoch 85/150
810/810 [==============================] - 167s 206ms/step - loss: 0.2843 - accuracy: 0.9144 - val_loss: 1.1604 - val
_accuracy: 0.7383
Epoch 86/150
```

In [207]:
```
plot_metrics(history_new_res)
```



And now we can see, that adding just one more step into the data augmentation makes it ~5% more accurate and greatly reduce the validation loss even on lower numbers of epochs (150 vs 200 with less augmentation). And it is really great! It is much easier to preprocess images, than try to optimize the number and order of layers, and more time effective than trying to fit NN more.

In [39]:
```
make_a_prediction(model_res_new,"h2.jpg")
```

Out[39]: 'The car on the image is most likely - HUMMER H2 SUT Crew Cab 2009'

Comment: perfect match!

In [40]:
```
make_a_prediction(model_res_new,"audi_q3.jpg")
```

Out[40]: 'The car on the image is most likely - Audi R8 Coupe 2012'

Comment: So, it understand that it is an Audi, however it is Q3 Sportback while R8 is a sport car. Not very accurate, I supposed there had been some Audi crossovers in the dataset

In [41]:
```
make_a_prediction(model_res_new,"Ferrari_SF90.jpg")
```

Out[41]: 'The car on the image is most likely - Ferrari 458 Italia Convertible 2012'

Comment: So, again, it is not 458 Italia, it is SF90, however, they are really quite similar

In [190]:
```
make_confusion_mtrx(model_res_new,val_ds,'conf_mtx.jpg')
```

Out[190]: <AxesSubplot:>