CarStyle

October 2, 2024

[1]:
```
import tensorflow as tf
import os
import cv2
import math
import json
import numpy as np
from matplotlib import pyplot as plt
from keras.applications import VGG19
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.metrics import Precision, Recall, SparseCategoricalAccuracy
```

[2]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        print(e)
```

```
Num GPUs Available:  1
1 Physical GPUs, 1 Logical GPUs
```

[3]:
```
base_dir = 'Styles'
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'valid')
test_dir = os.path.join(base_dir, 'test')

img_size = (224, 224)
batch_size = 32

train_data = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    image_size=img_size,
    batch_size=batch_size,
```

```
    label_mode='int',
    interpolation='bilinear'
)

val_data = tf.keras.utils.image_dataset_from_directory(
    val_dir,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='int',
    interpolation='bilinear'
)

test_data = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='int',
    interpolation='bilinear'
)
```

```
Found 5350 files belonging to 7 classes.
Found 1397 files belonging to 7 classes.
Found 802 files belonging to 7 classes.
```

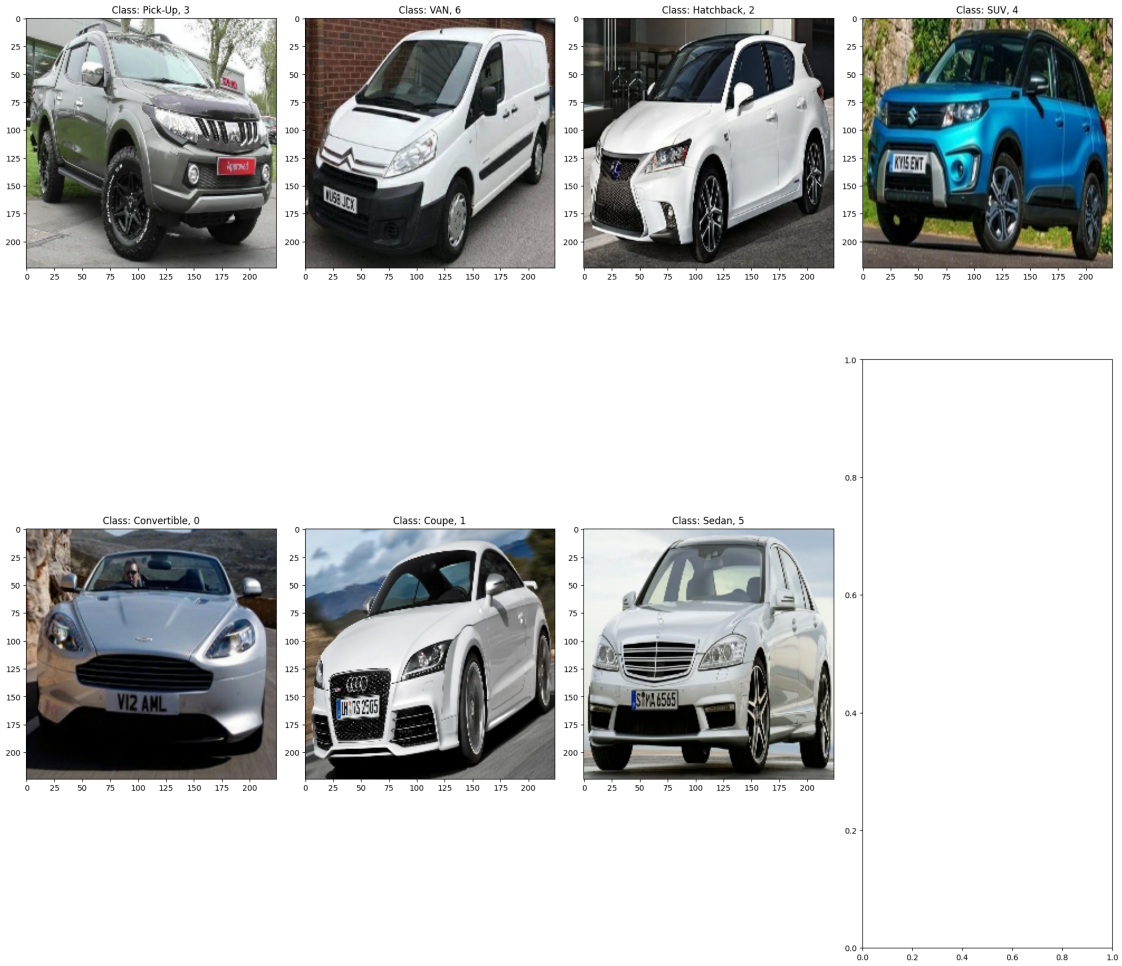
[4]:
```
class_names = train_data.class_names
print("Class names test:", class_names)

with open('CarStyle map.json', 'w') as f:
    json.dump(class_names, f)

data_iterator = train_data.as_numpy_iterator()
```

```
Class names test: ['Convertible', 'Coupe', 'Hatchback', 'Pick-Up', 'SUV',
'Sedan', 'VAN']
```

[5]:
```
batch = data_iterator.next()
num_classes = len(class_names)
```

[6]:
```
ncols = 4
nrows = math.ceil(num_classes / ncols)
fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(20, 20))

if nrows == 1:
    ax = ax.flatten()
elif ncols == 1:
    ax = ax.flatten()

plotted = set()
```

```
count = 0
while count < num_classes:
    batch = next(data_iterator)

    for idx, img in enumerate(batch[0]):
        label = batch[1][idx]
        if label not in plotted:
            ax_idx = count if nrows == 1 or ncols == 1 else (count // ncols,␣
 ↪count % ncols)
            ax[ax_idx].imshow(img.astype(int))
            ax[ax_idx].title.set_text(f"Class: {class_names[label]}, {label}")
            plotted.add(label)
            count += 1

        if count == num_classes:
            break

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



[7]:
```
base_model = VGG19(
    weights='imagenet',
    include_top=False,
    input_shape=(224, 224, 3)
)
base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 1s 0us/step
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808
```

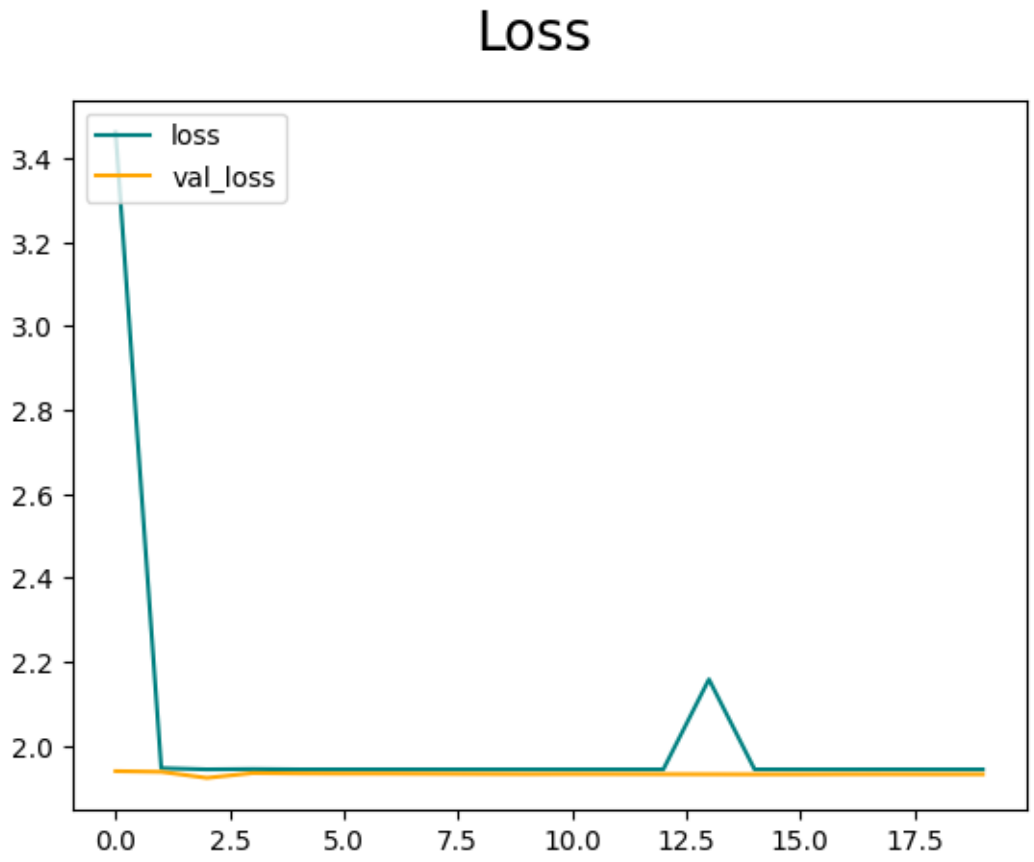
```
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

[8]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
output = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output)

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

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='logs')

hist = model.fit(train_data, epochs=20, validation_data=val_data,␣
 ↪callbacks=[tensorboard_callback])
```

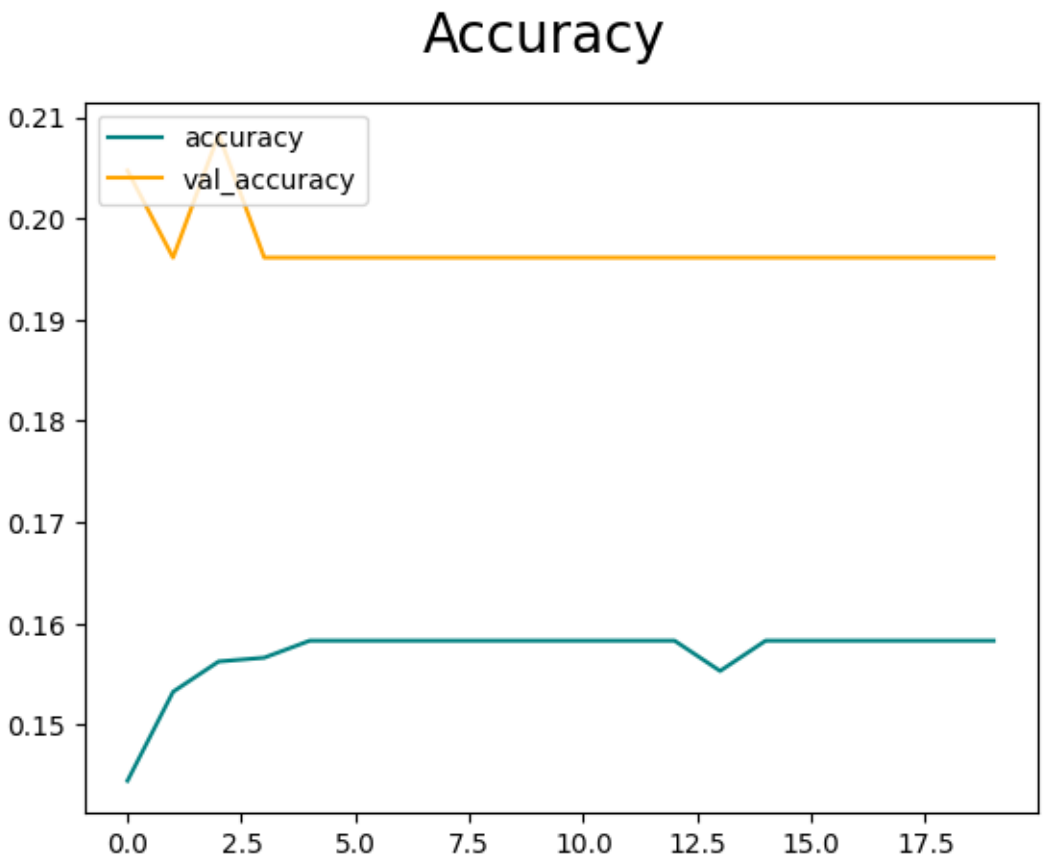
```
Epoch 1/20
168/168 [==============================] - 47s 219ms/step - loss: 3.4621 -
accuracy: 0.1445 - val_loss: 1.9399 - val_accuracy: 0.2047
Epoch 2/20
168/168 [==============================] - 35s 208ms/step - loss: 1.9470 -
accuracy: 0.1533 - val_loss: 1.9384 - val_accuracy: 0.1961
Epoch 3/20
168/168 [==============================] - 35s 210ms/step - loss: 1.9442 -
accuracy: 0.1563 - val_loss: 1.9239 - val_accuracy: 0.2083
Epoch 4/20
168/168 [==============================] - 36s 211ms/step - loss: 1.9449 -
accuracy: 0.1566 - val_loss: 1.9354 - val_accuracy: 0.1961
Epoch 5/20
168/168 [==============================] - 36s 211ms/step - loss: 1.9442 -
accuracy: 0.1583 - val_loss: 1.9345 - val_accuracy: 0.1961
```

```
Epoch 6/20
168/168 [==============================] - 34s 203ms/step - loss: 1.9442 -
accuracy: 0.1583 - val_loss: 1.9342 - val_accuracy: 0.1961
Epoch 7/20
168/168 [==============================] - 34s 201ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9341 - val_accuracy: 0.1961
Epoch 8/20
168/168 [==============================] - 33s 198ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9337 - val_accuracy: 0.1961
Epoch 9/20
168/168 [==============================] - 33s 196ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9333 - val_accuracy: 0.1961
Epoch 10/20
168/168 [==============================] - 33s 196ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9330 - val_accuracy: 0.1961
Epoch 11/20
168/168 [==============================] - 33s 197ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9332 - val_accuracy: 0.1961
Epoch 12/20
168/168 [==============================] - 33s 198ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9330 - val_accuracy: 0.1961
Epoch 13/20
168/168 [==============================] - 33s 199ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9328 - val_accuracy: 0.1961
Epoch 14/20
168/168 [==============================] - 34s 200ms/step - loss: 2.1586 -
accuracy: 0.1553 - val_loss: 1.9326 - val_accuracy: 0.1961
Epoch 15/20
168/168 [==============================] - 34s 200ms/step - loss: 1.9442 -
accuracy: 0.1583 - val_loss: 1.9324 - val_accuracy: 0.1961
Epoch 16/20
168/168 [==============================] - 34s 200ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9325 - val_accuracy: 0.1961
Epoch 17/20
168/168 [==============================] - 34s 200ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9327 - val_accuracy: 0.1961
Epoch 18/20
168/168 [==============================] - 34s 202ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9328 - val_accuracy: 0.1961
Epoch 19/20
168/168 [==============================] - 34s 200ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9326 - val_accuracy: 0.1961
Epoch 20/20
168/168 [==============================] - 34s 199ms/step - loss: 1.9441 -
accuracy: 0.1583 - val_loss: 1.9326 - val_accuracy: 0.1961
```

[9]:
```
fig = plt.figure()
plt .plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



[10]:
```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



[11]:
```
pre = Precision()
re = Recall()
acc = SparseCategoricalAccuracy()
```

[12]:
```
for batch in test_data.as_numpy_iterator():
    X, y = batch
    yhat = model.predict(X)

    yhat_classes = tf.argmax(yhat, axis=1)

    pre.update_state(y, yhat_classes)
    re.update_state(y, yhat_classes)
    acc.update_state(y, yhat)
```

```
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
```

```
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 386ms/step
```

[13]:
```
print(f"Precision: {pre.result().numpy() * 100 : .2f}%")
print(f"Recall: {re.result().numpy() * 100 : .2f}%")
print(f"Accuracy: {acc.result().numpy() * 100 : .2f}%")
```

```
Precision:  78.30%
Recall:  100.00%
Accuracy:  15.59%
```

[14]:
```
img = cv2.imread('Styles/test/Hatchback/8_jpg.rf.
 ↪c314c1d6777942876503fa1482c82240.jpg')

img_resized = cv2.resize(img, img_size)
img_expanded = np.expand_dims(img_resized, axis=0)

yhat = model.predict(img_expanded)
predicted_class = tf.argmax(yhat, axis=1).numpy()[0]

plt.imshow(img)
plt.title(f'Predicted class: {predicted_class}')
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 333ms/step
```

Predicted class: 4

[15]:
```
print(f'Predicted class is: {class_names[predicted_class]}')
for idx, prob in enumerate(yhat[0]):
    print(f"Model probability for {class_names[idx]} is {prob * 100:.2f}%")
```

```
Predicted class is: SUV
Model probability for Convertible is 15.18%
Model probability for Coupe is 13.50%
Model probability for Hatchback is 13.01%
Model probability for Pick-Up is 14.21%
Model probability for SUV is 15.87%
Model probability for Sedan is 13.66%
Model probability for VAN is 14.56%
```

[16]:
```
model_file_name = f"CarStyle{acc.result().numpy() * 100 : .2f}% VGG19.h5"
model.save(os.path.join('CarBackEnd/models/CarStyles', model_file_name))
```

[ ]:
```

```