## Classification of Cars with Machine Learning

```
# This code installs and sets up the fastbook library as a dependency.

! [ -e /content ] && pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 719.8/719.8 kB 11.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 22.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.0/7.0 MB 75.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 468.7/468.7 kB 29.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 18.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.9/132.9 kB 12.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 kB 9.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 200.1/200.1 kB 11.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.2/212.2 kB 9.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 32.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.2/114.2 kB 1.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.8/158.8 kB 8.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 269.3/269.3 kB 7.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 39.1 MB/s eta 0:00:00
Mounted at /content/gdrive
```

```
# Import dependencies

from fastbook import *
from fastai.vision.widgets import *


# Setting key

key = os.environ.get('AZURE_SEARCH_KEY', '74f7dde4575e4799800cae9dbb7d6e6f')


# Search Images

search_images_bing
```

```
<function fastbook.search_images_bing(key, term, min_sz=128, max_images=150)>
```

```
# Search image of real cars

results = search_images_bing(key, 'carros')
ims = results.attrgot('contentUrl')
len(ims)
```

```
150
```

```
# Getting image from yellow car

ims = ['https://images.cdn.circlesix.co/image/1/640/0/uploads/articles/dsc_2049_60809-564c6f63611b8.jpg']


# Save the image

dest = 'images/yellow_car.jpg'
download_url(ims[0], dest)
```

```
-819200.00% [8192/-1 00:00<00:00]
-1638400.00% [16384/-1 00:00<00:00]
-2457600.00% [24576/-1 00:00<00:00]
-3276800.00% [32768/-1 00:00<00:00]
-4096000.00% [40960/-1 00:00<00:00]
-4915200.00% [49152/-1 00:00<00:00]
-5734400.00% [57344/-1 00:00<00:00]
-6553600.00% [65536/-1 00:00<00:00]
-7372800.00% [73728/-1 00:00<00:00]
Path('images/yellow_car.jpg')
```

```
# Open and show the image

im = Image.open(dest)
im.to_thumb(256,256)
```

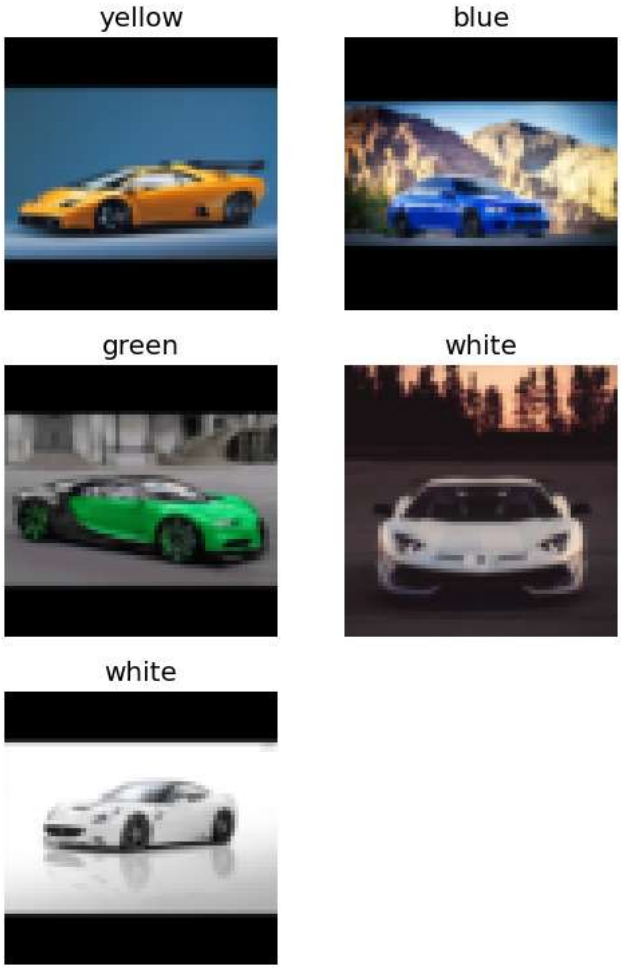
```
# Setting avaliables car colors

cars_colors = 'black','white','blue', 'yellow','green'
path = Path('carros')


# Get image from each cars_colors and download it

if not path.exists():
    path.mkdir()
    for o in cars_colors:
        dest = (path/o)
        dest.mkdir(exist_ok=True)
        results = search_images_bing(key, f'{o} carros')
        download_images(dest, urls=results.attrgot('contentUrl'))


# Gets all the image files within the specified directory path and assigns them to the variable fns

fns = get_image_files(path)
fns
```

```
(#705) [Path('carros/black/987f4963-b807-45fe-824a-a04e6e9b9eab.jpg'),Path('carros/black/ebe664e3-77d6-4981-9c99-
5af1d893568c.jpg'),Path('carros/black/7baaf5c0-9b25-445d-a8ad-54710156cf48.jpg'),Path('carros/black/8dc4cd2e-13af-434c-b5ec-
6d3b4a4432c0.jpg'),Path('carros/black/ae4f9f59-047f-45a8-a733-9c15bd3652b7.jpg'),Path('carros/black/1998e741-cabd-44de-a4e3-
28ad5ef3e5f7.jpg'),Path('carros/black/600e316f-9119-42a6-8b37-f58bd7af7315.jpg'),Path('carros/black/67575ac7-c84d-4d58-827d-
1838b4bb0905.jpg'),Path('carros/black/a1a04751-e4cd-4a68-b682-b29a02bc152a.jpg'),Path('carros/black/7f0f66be-eced-45a6-b88d-
0280569edd67.jpg')...]
```

```
# Prepare the data for training a model on images of cars

cars = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=Resize(128))


# This code creates a dataloader object dls from the cars DataBlock using the path specified earlier as the source of the data.

dls = cars.dataloaders(path)


# Will display a batch of images from the validation set of the dataloaders object created from the cars DataBlock.

dls.valid.show_batch()
```

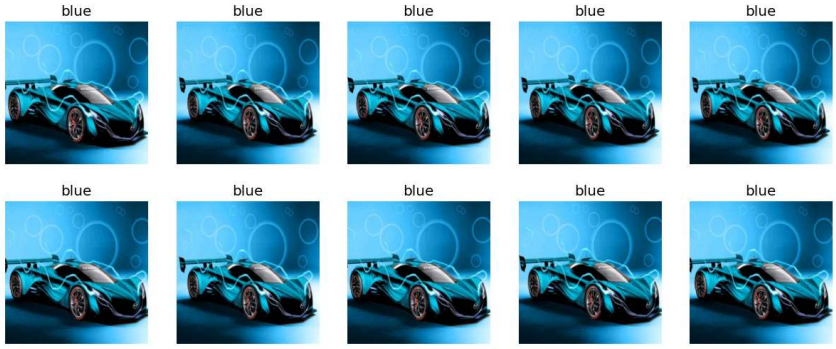

```
# This code block resizes the images in the cars datablock to a fixed size of 128 pixels using the Resize transformation with ResizeMeth
# creates a new dataloader with the resized images, and displays a batch of 4 unique images from the validation set with a maximum of 1

cars = cars.new(item_tfms=Resize(128, ResizeMethod.Squish))
dls = cars.dataloaders(path)
dls.valid.show_batch(max_n=4, nrows=1, unique=True)
```


```
# Resizes the images to have a height and width of 64 pixels using padding to maintain the aspect ratio of the images
# and sets the padding mode to 'zeros'. Then it creates a new dataloader object based on the modified dataset and shows a batch of four

cars = cars.new(item_tfms=Resize(64, ResizeMethod.Pad, pad_mode='zeros'))
dls = cars.dataloaders(path)
dls.valid.show_batch(max_n=5, nrows=3)
```




```
# This code randomly crops images in the training set using RandomResizedCrop with a minimum scale of 0.89 and displays a batch of 4 uni
```

```
cars = cars.new(item_tfms=RandomResizedCrop(224, min_scale=0.89))
dls = cars.dataloaders(path)
dls.train.show_batch(max_n=10, nrows=2, unique=True)
```




```
# This code sets up a convolutional neural network using the ResNet18 architecture
# and fine-tunes it on the given dataloaders for 4 epochs. The performance of the model is evaluated using the error_rate metric.
learn = cnn_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(4)
```

```
/usr/local/lib/python3.9/dist-packages/fastai/vision/learner.py:288: UserWarning:
  warn("`cnn_learner` has been renamed to `vision_learner` -- please update your c
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:208: UserWarni
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:223: UserWarni
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/
100%|██████████| 44.7M/44.7M [00:00<00:00, 206MB/s]
```

| epoch | train_loss | valid_loss | error_rate | time |
|---|---|---|---|---|
| 0 | 2.506444 | 0.805031 | 0.294964 | 00:54 |

| epoch | train_loss | valid_loss | error_rate | time |
|---|---|---|---|---|
| 0 | 0.780268 | 0.438975 | 0.107914 | 01:00 |
| 1 | 0.488997 | 0.343556 | 0.079137 | 00:53 |
| 2 | 0.330993 | 0.346030 | 0.064748 | 00:55 |
| 3 | 0.242773 | 0.353371 | 0.071942 | 00:52 |

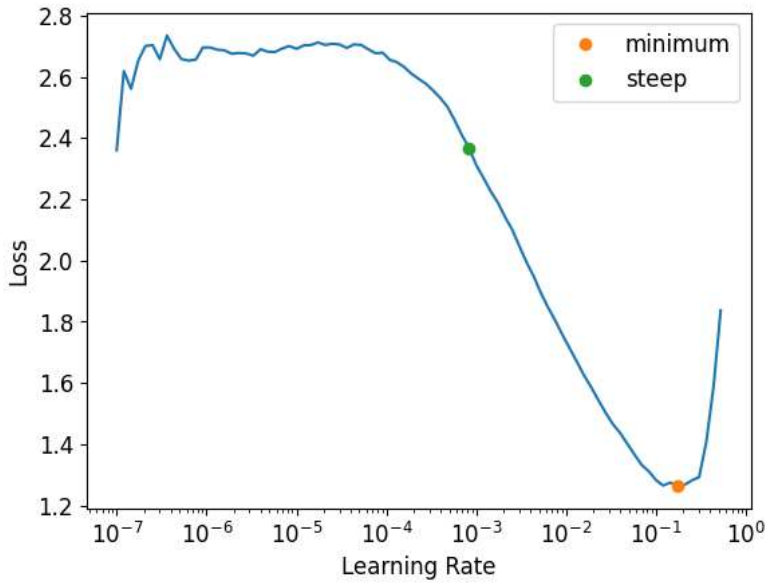
```
#·The·code·creates·a·convolutional·neural·network·learner·object·learn·using·a·ResNet34·model·architecture·and·the·dataloaders·dls.
#·It·then·fine-tunes·the·pre-trained·model·for·4·epochs·with·a·base·learning·rate·of·0.1,·using·the·fine_tune()·method.·The·metric·used·

learn·=·cnn_learner(dls,·resnet34,·metrics=error_rate)
learn.fine_tune(4,·base_lr=·0.1)
```

```
/usr/local/lib/python3.9/dist-packages/torchvision/models/_utils.py:223: UserWarni
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/
100%|██████████| 83.3M/83.3M [00:00<00:00, 95.9MB/s]
```

| epoch | train_loss | valid_loss | error_rate | time |
|---|---|---|---|---|
| 0 | 1.236874 | 6.496347 | 0.366906 | 00:54 |

| epoch | train_loss | valid_loss | error_rate | time |
|---|---|---|---|---|
| 0 | 1.116350 | 1469.571777 | 0.820144 | 00:52 |
| 1 | 1.178550 | 76.311096 | 0.388489 | 00:53 |
| 2 | 0.907831 | 20.838860 | 0.503597 | 00:53 |
| 3 | 0.742382 | 1.238460 | 0.143885 | 00:52 |

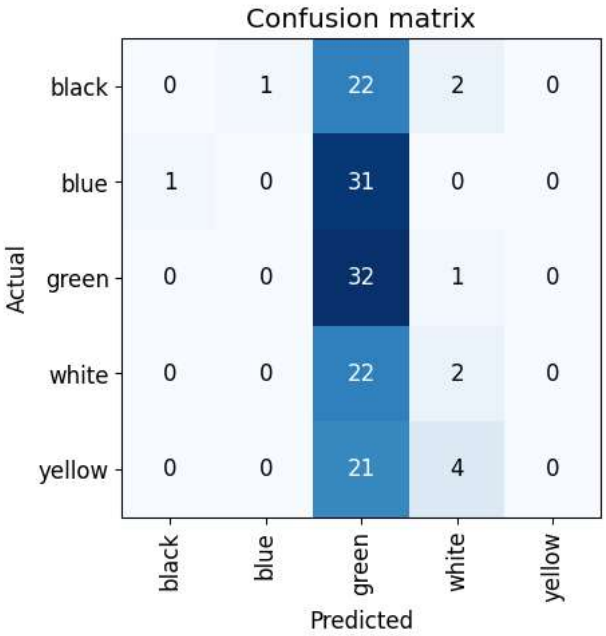
```
# The code defines a cnn_learner object with a ResNet34 architecture and the error_rate metric.
# it then calls the fine_tune method to train the model for 4 epochs with a default learning rate.

learn = cnn_learner(dls, resnet34, metrics=error_rate)
md = learn.lr_find(suggest_funcs=(minimum, steep))
```




```
# Print

print(f"Minimum/10: {md.minimum:.2e}, steepest point: {md.steep:.2e}")
```

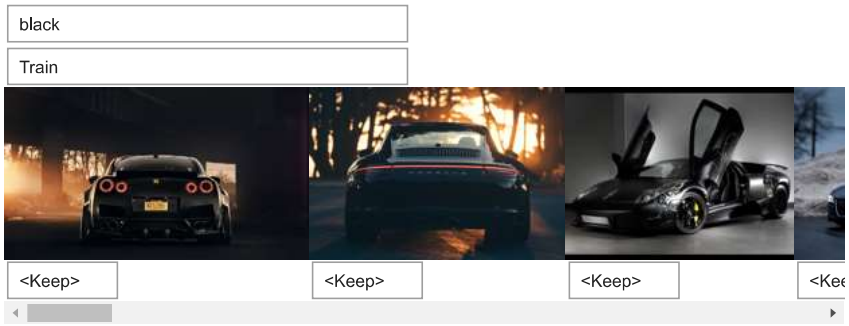
```
Minimum/10: 1.74e-02, steepest point: 8.32e-04
```

```
# creates an instance of the ClassificationInterpretation class from the cnn_learner model
# and uses it to plot a confusion matrix, which is a visual representation of the performance of the model in classifying the validation
# showing the number of true positive, false positive, true negative, and false negative predictions for each class.

interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix()
```




```
#  generates a plot of the top losses in the validation set, showing the image with highest loss,
# the predicted label and the true label. In this case, the plot shows the top 3 losses, with all images in the same row.

interp.plot_top_losses(3, nrows=1)
```


```
# Creates an instance of the ImageClassifierCleaner class with the specified model.

cleaner = ImageClassifierCleaner(learn)
cleaner
```




## Deploy

```
# Export

learn.export()


# Export the file

path = Path()
path.ls(file_exts='.pkl')

     (#1) [Path('export.pkl')]


# Load model

learn_inf = load_learner(path/'export.pkl')


# Chance

learn_inf.predict('images/yellow_car.jpg')

     ('white',
      tensor(3),
      tensor([2.8685e-05, 3.8698e-03, 1.6642e-01, 8.2601e-01, 3.6729e-03]))


# Options

learn_inf.dls.vocab

     ['black', 'blue', 'green', 'white', 'yellow']


# The code creates a button to upload an image file, a button to classify the image, an output widget to display the image
# and a label widget to show the prediction and probability of the classification.
# The on_click_classify function is called when the classify button is clicked, which reads the uploaded image
# displays it in the output widget, and performs a prediction using the trained model, which is then displayed in the label widget.

btn_upload = widgets.FileUpload()
btn_run = widgets.Button(description='Classify')
out_pl = widgets.Output()
lbl_pred = widgets.Label()

def on_click_classify(change):
    img = PILImage.create(btn_upload.data[-1])
    out_pl.clear_output()
    with out_pl: display(img.to_thumb(128,128))
    pred,pred_idx,probs = model_inf.predict(img)
    lbl_pred.value = f'Prediction: {pred}; Probability: {probs[pred_idx]:.04f}'
```

```
btn_run.on_click(on_click_classify)


# Fake the image

btn_upload = SimpleNamespace(data = ['images/yellow_car.jpg'])


# Create

img = PILImage.create(btn_upload.data[-1])


# out_pl is an Output widget that can be used to display output from other widgets.
# Here, the code is clearing the output and displaying the thumbnail of an image of size 256x256 in the out_pl widget.
# However, since there is no image variable defined in this code, it will throw an error.

out_pl = widgets.Output()
out_pl.clear_output()
with out_pl: display(img.to_thumb(256,256))
out_pl
```

```
# This line of code uses the predict method of a learner object to make predictions on an input image

pred,pred_idx,probs = learn_inf.predict(img)



# The code creates a label widget and sets its value to a string containing the predicted label and probability for an image classificat

lbl_pred = widgets.Label()
lbl_pred.value = f'Prediction: {pred}; Probability: {probs[pred_idx]:.04f}'
lbl_pred
```

Prediction: white; Probability: 0.8260

```
# Run the classify

btn_run = widgets.Button(description='Classify')
btn_run
```

Classify

```
# Classify function
def on_click_classify(change):
    img = PILImage.create(btn_upload.data[-1])
    out_pl.clear_output()
    with out_pl: display(img.to_thumb(128,128))
    pred,pred_idx,probs = learn_inf.predict(img)
    lbl_pred.value = f'Prediction: {pred}; Probability: {probs[pred_idx]:.04f}'

btn_run.on_click(on_click_classify)


# Widget of file upload

btn_upload = widgets.FileUpload()


# Widget showing the chance
VBox([widgets.Label('Select the car from your computer'),
      btn_upload, btn_run, out_pl, lbl_pred])
```

Select the car from your computer

Upload (0)

Classify

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Prediction: white, Probability: 0.8280

✓ 3s conclusão: 16:57