demo

November 11, 2023

[14]:
```
from torchvision import models
import sys, random, os
sys.path.append('./src/utils')
sys.path.append('./src/visualize/')
import src.step3_test_on_SPASL_v1.MODULE_test as mod_test
import src.visualize.MODULE_visualize as mod_vis
import src.utils.MODULE_utils as mod_utils
import pandas as pd
```

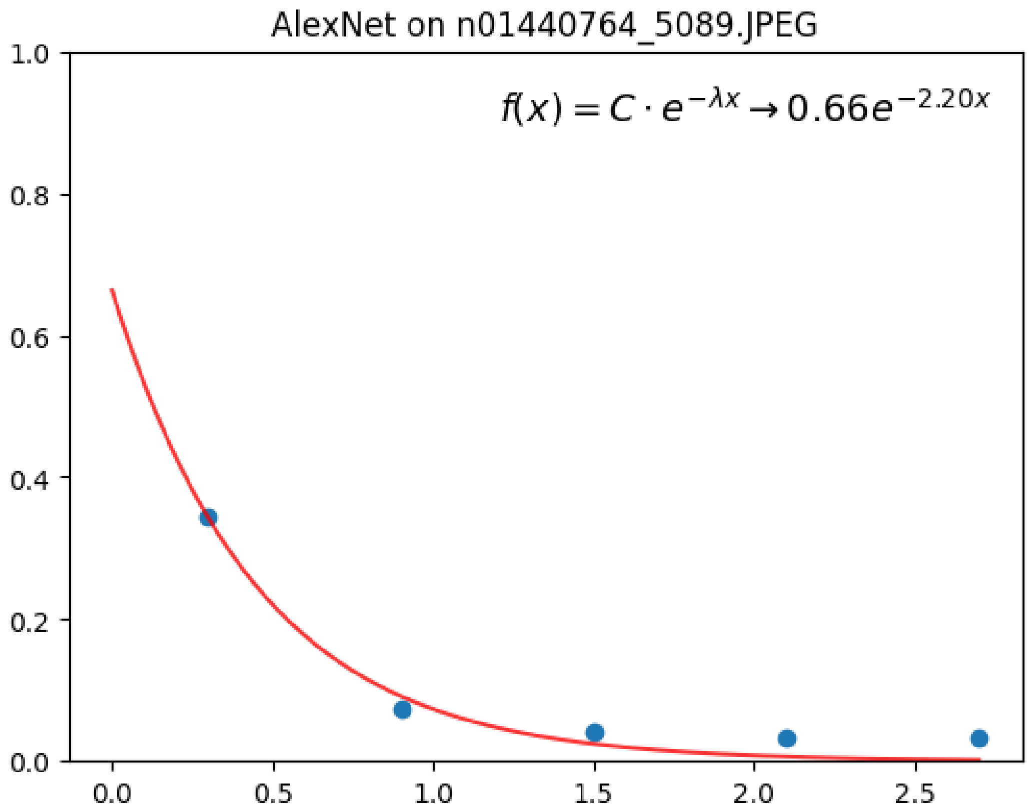
0.1 Experiment 1: Curve fitting on top-5 prediction probabilities

To quantify the prediction confidence of a model on an image, we first map the top-5 predictin probabilities (p_i, where $p_i \in (0,1)$) to five points ((x_i, p_i), where $x_i \in \{0.3, 0.9, 1.5, 2.1, 2.7\}$) in quadrant I. Then, we fit a curve on these five points. The best one we found is a scaled exponential curve: $f(x) = C \cdot e^{-\lambda x}$.

Below, we demonstrate the process of fitting the curve on top-5 prediction probabilities for a model on an image. Predictions provided here are from AlexNet and MaxVit on 500 n01440764 images. We also compare the fitted curves on the same image based on two models' predictions.

[15]:
```
# The path of the IW table of AlexNet on 500 n01440764 images
model_name1 = 'AlexNet'
IW_path1 = f'./demo/exp_materials/{model_name1}_n01440764_after_AA.csv'
IW_df1 = pd.read_csv(IW_path1)

# Choose a random image
img_idx = random.randint(0, len(IW_df1)-1)

# Extract 5 predictions from AlexNet on it
prediction_probabilities1 = list(IW_df1.iloc[img_idx, 1:6])

# Plot the curve
mod_vis.plot_curve_and_points(prediction_probabilities1, f'AlexNet on {IW_df1.
↪iloc[img_idx, 0]}')
```

```
Fitting a curve on [(0.3, 0.34554), (0.9, 0.07425), (1.5, 0.04234), (2.1,
0.03201), (2.7, 0.03187)]
C = 0.66342225871427,   = 2.200553317838107
```

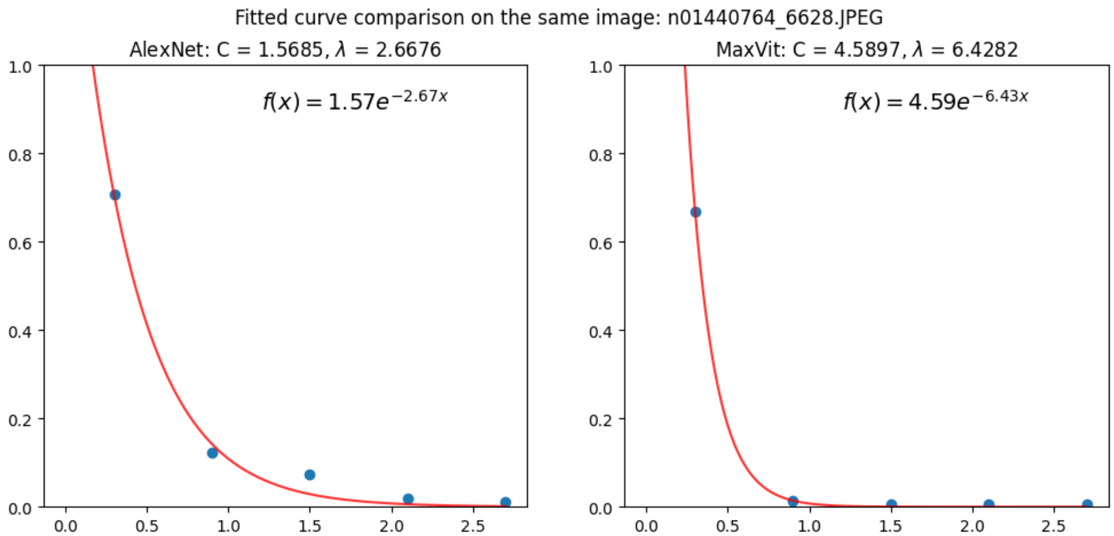


[16]:

```python
# We compare the predictin proabilities of a more recent model, MaxVit, with␣
↪AlexNet on the same 500 images
model_name2 = 'MaxVit'
IW_path2 = f'./demo/exp_materials/{model_name2}_n01440764_after_AA.csv'
IW_df2 = pd.read_csv(IW_path2)

# Choose a random image
img_idx = random.randint(0, 499)

# Extract 5 prediction probabilities of AlexNet and MaxVit on it
prediction_probabilities1 = list(IW_df1.iloc[img_idx, 1:6]) # AlexNet's␣
↪prediction probabilities
prediction_probabilities2 = list(IW_df2.iloc[img_idx, 1:6]) # MaxVit's␣
↪prediction probabilities

# Ensure their predictions are about the same image
assert IW_df1.iloc[img_idx, 0] == IW_df2.iloc[img_idx, 0], 'Please make sure␣
↪the images are the same.'
image_name = IW_df1.iloc[img_idx, 0]
```

```
# Plot the fitted curve and compare the parameters
mod_vis.compare_curves_on_the_same_image(prediction_probabilities1,␣
↪model_name1, prediction_probabilities2, model_name2, image_name)
```

```
Fitting a curve on AlexNet's prediction probabilities: [0.7067, 0.12205,
0.07345, 0.01988, 0.01158]
Fitting a curve on MaxVit's prediction probabilities: [0.66721, 0.01381,
0.00695, 0.00644, 0.0064]
```

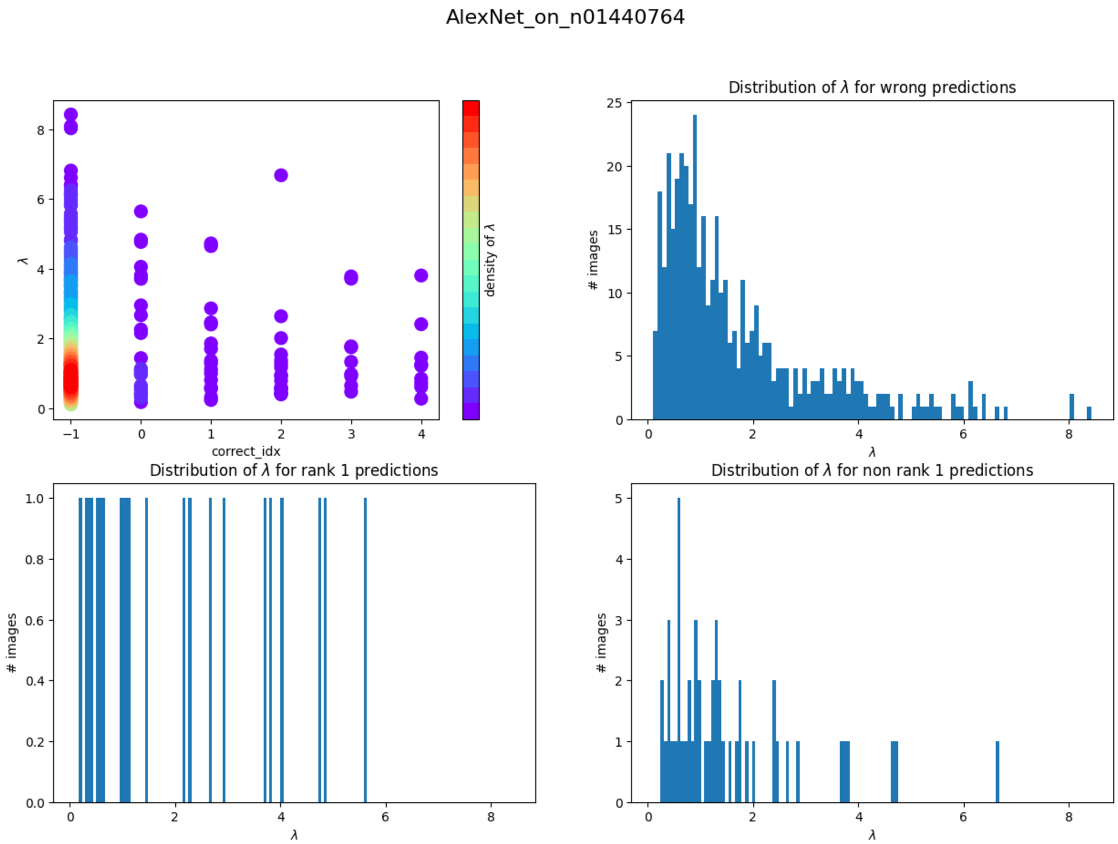


0.2 Experiment 2: λ distribution of a model on a class of images

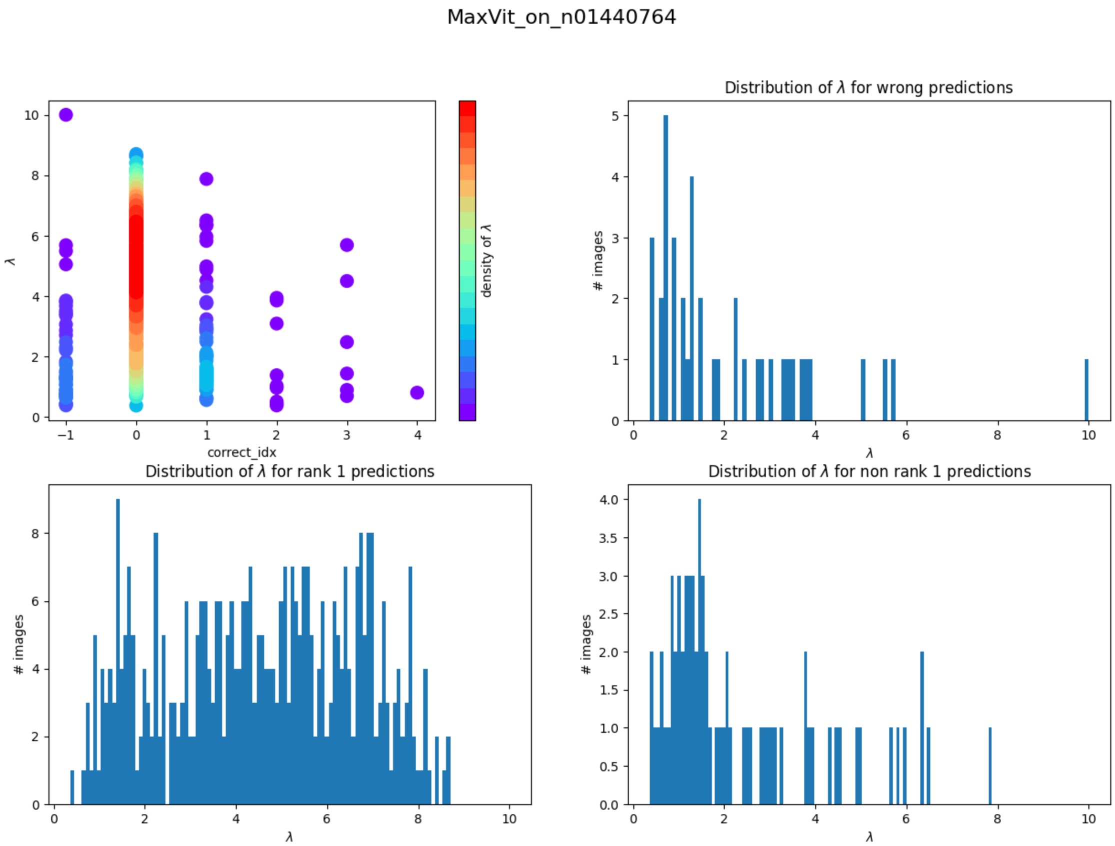
After gathering 5 predictions and probabilities of a model on each image in a class, we determine the λ of the fitted exponential curve and correct_idx. A larger value of λ indicates higher confidence. correct_idx $= -1$ indicates the correct prediction is not among the 5 predictions. correct_idx $= 0$ suggests that the prediction (whose rank is 1) with the highest probabilty is the correct one. The other possible values of correct_idx are $1, 2, 3, 4$. They are categorized as "non rank 1" predictions.

In this experiment, we show the distribution of λ of AlexNet and MaxVit. Clear λ distribution difference can be observed between two given models.

[17]:
```
# We provide two IW tables: AlexNet and MaxVit on 500 n014407864 images after␣
↪applying adversarial distortion
# IW table location: './demo/AlexNet(or MaxVit)_n01440764_after_AA.csv'

model_name = 'AlexNet'
class_name = 'n01440764'
csv_path = f'./demo/exp_materials/{model_name}_{class_name}_after_AA.csv'
mod_vis.plot_single_prediction_and_lambda(csv_path, model_name = model_name,␣
↪class_name = class_name, show = True, save_path = None)
```



[18]:
```python
model_name = 'MaxVit'
csv_path = f'./demo/exp_materials/{model_name}_{class_name}_after_AA.csv'
mod_vis.plot_single_prediction_and_lambda(csv_path, model_name = model_name,
 ↪class_name = class_name, show = True, save_path = None)
```



After extensive experiments and observations, we provide an empirical λ value to quantify the boundary between confident and not confident predictions of a model on an image. In this way, we can determine the prediction pattern of a model on an image into one of the six identified patterns: Type-I/II/III/IV and Type-I/II-NRO.

	I	**I-NRO**	**II**	**II-NRO**	**III**	**IV**
λ	> 4	> 4	< 4	< 4	< 4	> 4
correct_idx	0	> 1	0	> 1	-1	-1

We also define that if an image tends to cause Type-x prediction pattern for a lot of models, this image is called a Type-x image. The 500 n014407864 images listed in the table are Type-I images, determined by cross searching. In other words, they are the easiest 500 out of 1300 images originally provided by ImageNet-1K in this class.

0.3 Experiment 3: Cross search

[19]:
```
# We test all ICAC models on each image and compute the lambda value of the fitted curve, and the correct_idx
# They are usually stored in difference csv files for a clean look
```

```
# Location of SPASL-general's IW_table of lambda values. This benchmark variant␣
↪has an ICAC size of 80
# Total # colums = 81
IW_table_lambda = pd.read_csv('./demo/exp_materials/
↪IW_table_general_lambda_n01440764.csv')

# We display the first 5 rows
IW_table_lambda.head(5)
```

[19]:

```
                filename  regnet_y_800mf  regnet_x_16gf  vgg13   vgg19  ViTB32  \
0  n01440764_10026.JPEG           1.289          8.168  0.404   0.346   9.068
1  n01440764_10027.JPEG           9.372         11.489  9.311  11.189  11.175
2  n01440764_10029.JPEG           3.746          3.090  2.173   0.486  11.279
3  n01440764_10040.JPEG           9.458          3.853  0.583   0.822   9.921
4  n01440764_10042.JPEG           1.370          1.733  2.576   0.878   1.412

   EfficientNetV2_l  EfficientNetb2  DenseNet169  regnet_x_800mf  …  \
0             9.662           4.077       11.509           3.152  …
1             6.650           9.309       11.513           9.449  …
2             8.079           6.363        5.685           1.034  …
3             8.125          10.375        1.942           1.034  …
4             4.247           0.649        6.773           3.517  …

   squeezenet1_1  regnet_y_128gf  MobileNet_v3_small  DenseNet161  GoogLeNet  \
0          0.544           7.542               0.961       11.510      1.296
1          0.630           8.416               0.312       11.513      9.027
2          0.778           9.771               5.816        2.375      9.762
3          0.746           7.537               8.397        1.751      1.557
4          0.698           6.147               0.524       10.213      2.889

   EfficientNetb0  regnet_y_3_2gf  vgg11  shufflenet_v2_x2_0  ViTL32
0           7.735           8.984  0.375               3.138   8.158
1           8.172           9.506  7.807              11.421   6.290
2           4.695           8.728  0.469               0.343   9.889
3           7.126           9.819  2.462               0.275   4.221
4           1.398           1.845  1.106               1.884   2.715

[5 rows x 81 columns]
```

[20]:

```
# If a SPASL benchmark variant has a smaller ICAC, the IW_table construction␣
↪will be faster.

# Location of SPASL-vit's IW_table of lambda values. This benchmark variant has␣
↪an ICAC size of 12
# Total # colums = 13
```

```python
IW_table_lambda_vit = pd.read_csv('./demo/exp_materials/
↪IW_table_vit_lambda_n01440764.csv')

# We display the first 5 rows
IW_table_lambda_vit.head(5)
```

[20]:

	filename	swin_b	swin_s	swin_t	ViTB16	ViTB32	ViTH14
0	n01440764_10026.JPEG	3.672	4.651	3.062	4.970	9.068	3.178
1	n01440764_10027.JPEG	4.373	9.305	2.625	4.313	11.175	0.506
2	n01440764_10029.JPEG	7.325	6.064	6.091	4.196	11.279	0.633
3	n01440764_10040.JPEG	8.052	7.828	6.460	0.968	9.921	0.480
4	n01440764_10042.JPEG	4.208	5.689	3.152	0.410	1.412	0.829

	ViTL16	ViTL32	MaxVit	swin_v2_b	swin_v2_s	swin_v2_t
0	9.974	8.158	8.105	3.265	0.929	2.385
1	10.569	6.290	9.122	9.131	9.098	8.351
2	11.015	9.889	8.651	6.745	4.817	6.162
3	9.077	4.221	7.660	4.630	5.637	6.744
4	5.548	2.715	1.648	3.670	1.305	0.799

[21]:

```python
# Back to SPASL-general. Its IW_table of correct_idx also has 81 columns
IW_table_correct_id = pd.read_csv('./demo/exp_materials/
↪IW_table_general_correct_idx_n01440764.csv')

# We display the first 5 rows
IW_table_correct_id.head(5)
```

[21]:

	filename	regnet_y_800mf	regnet_x_16gf	vgg13	vgg19	ViTB32
0	n01440764_10026.JPEG	0	0	0	2	0
1	n01440764_10027.JPEG	0	0	0	0	0
2	n01440764_10029.JPEG	0	0	4	1	0
3	n01440764_10040.JPEG	0	0	-1	0	0
4	n01440764_10042.JPEG	4	1	4	-1	1

	EfficientNetV2_l	EfficientNetb2	DenseNet169	regnet_x_800mf	…
0	0	0	0	0	…
1	0	0	0	0	…
2	0	0	0	0	…
3	0	0	0	0	…
4	-1	0	4	2	…

	squeezenet1_1	regnet_y_128gf	MobileNet_v3_small	DenseNet161	GoogLeNet
0	-1	0	-1	0	0
1	-1	0	1	0	0
2	-1	0	0	0	0
3	-1	0	0	0	0
4	-1	-1	-1	3	4

```
   EfficientNetb0  regnet_y_3_2gf  vgg11  shufflenet_v2_x2_0  ViTL32
0               0               0     -1                  -1       0
1               0               0      0                   0       0
2               0               0      4                  -1       0
3               0               0     -1                  -1       0
4               4               4     -1                  -1       1

[5 rows x 81 columns]
```

[22]:
```python
# We calculate some statistics of lambda values, and count the number of correct (rank-1, non-rank-1), and wrong predictions.
# They are stored at an IW_summary table
IW_table_summary_path = './demo/exp_materials/IW_table_general_summary_n01440764.csv'
IW_table_summary = pd.read_csv(IW_table_summary_path)

# We display the first 5 rows
IW_table_summary.head(5)
```

[22]:
```
              filename   lamb_Q3  lamb_Q2  lamb_Q1  R1_Count  Wrong_Count  \
0  n01440764_10026.JPEG   8.33450   5.4125  2.97350        66            8
1  n01440764_10027.JPEG  11.43125   9.3100  7.52775        73            5
2  n01440764_10029.JPEG   7.85100   4.0755  1.08275        67            7
3  n01440764_10040.JPEG   7.91400   4.2045  1.60750        66            8
4  n01440764_10042.JPEG   3.52225   1.6905  1.03300         5           25

   Non_R1_Count
0             6
1             2
2             6
3             6
4            50
```

[23]:
```python
# Cross search on conditions of "R1_Count" (primary) and "lambda_Q3" for 500 Type-I images
Type_I_500_n01440764 = mod_utils.cross_search_for_top_n_images(IW_table_summary_path, 500, 'I')

# Display the top-10 images with the most R1_Count and lambda_Q3, thus, most models produce Type-I prediction patterns on them.
Type_I_500_n01440764.head(10)
```

[23]:
```
             filename   lamb_Q3  lamb_Q2  lamb_Q1  R1_Count  Wrong_Count  \
0  n01440764_8045.JPEG  10.23250   8.1615  4.74700        80            0
1  n01440764_6831.JPEG  11.00075   8.6765  6.46950        79            1
2  n01440764_9191.JPEG   7.91300   4.9090  3.19300        79            1
```

```
3  n01440764_7913.JPEG  10.62100   7.8840  6.01825        79             1
4  n01440764_6641.JPEG  10.77950   8.9105  6.97450        79             1
5  n01440764_6672.JPEG   9.99450   8.4150  5.80050        78             0
6  n01440764_3867.JPEG   8.43600   4.2165  2.72125        78             1
7  n01440764_5990.JPEG   9.61925   7.6795  5.63725        78             1
8  n01440764_8246.JPEG   7.23550   4.7280  3.15925        78             1
9  n01440764_6628.JPEG   7.84175   4.9880  4.25875        78             1

   Non_R1_Count
0             0
1             0
2             0
3             0
4             0
5             2
6             1
7             1
8             1
9             1
```

[24]:
```
# Display the last 10 images of these 500 Type-I images
# It proves that they are also easy: 68 out of 80 can still provide a Type-I␣
↪prediction pattern
Type_I_500_n01440764.tail(10)
```

[24]:
```
                filename   lamb_Q3  lamb_Q2  lamb_Q1  R1_Count  Wrong_Count  \
491  n01440764_13316.JPEG   7.61200   5.6365  3.09725        68            4
492   n01440764_3259.JPEG   7.12225   2.9760  1.52300        68            4
493  n01440764_19777.JPEG   7.69550   4.1125  1.94450        68            6
494  n01440764_22948.JPEG   7.89800   4.3445  2.79725        68            8
495  n01440764_11974.JPEG   8.86025   5.9685  2.57450        68            5
496  n01440764_11314.JPEG   7.48675   3.8660  1.45575        68            6
497   n01440764_2939.JPEG   6.81525   2.3260  0.58450        68            9
498   n01440764_1154.JPEG   7.92975   5.4990  2.96125        68            8
499   n01440764_6719.JPEG  10.07775   7.2015  4.21625        68            6
500   n01440764_2563.JPEG   8.63175   4.7245  2.39400        68            5

     Non_R1_Count
491             8
492             8
493             6
494             4
495             7
496             6
497             3
498             4
499             6
```

```
500            7
```

0.3.1 Comments

1. The image-wise (IW) table construction is based on the test results of ICAC models. A smaller ICAC can boost the IW table construction speed, hence preferred.

2. For each of 1 000 classes in ImageNet-1K, we generate a corresponding IW table, which initially gathers top-5 predictions and probabilites, and later λ and correct_idx information. According to this IW table, we determine 100 Type-IV images, and 500 Type-I images by cross search. Type-IV images are the most difficult ones in each class, and they are included as the SuperHard (SH) component. For the rest four components, we apply modifications to 500 easiest images and cross search for 100 most difficulty modified images that fool most ICAC models. These images are included as the corresponding SPASP component.

3. Reason for constructing SH based on ImageNet-1K training set, rather than test set, is that all the pretrained models have been trained on the training set for a lot of times. In other words, they have seen all the training images for many times. If most of them still cannot provide a correct prediction, the only reason is that the image is very difficult. As human beings, we may not be able to explain why some images fool so many models: they look fine to us. This explains why the interpretability studies are important. And our benchmark will promost such studies. Obtaining the final SPASL_v1, we do observe some mislabeled images that pretends to fool a lot of models. We overlook this mislabeling issue for now.

4. Reason for constructing the other four components based on easiest images in each class is that we want to make sure that an image converted from Type-I to Type-IV is only because the modification, not the image being difficult at the first place. In this way, we can evaluate the effectiveness of a modification, or attack, by studying the rates of such Type-I to Type-IV conversion. Reversely, we can evaluate restoration methods.

The modifications are detailed explained in the paper and we will skip the experiment of modified example generation.

0.4 Experiment 4: Test a model on SPASL_demo

SPASL_demo is a tiny subsets (10 classes, $\approx$ 50 MB) of SPASL_v1 (1 000 classes, $\approx$ 6 GB), only for demo purpose. This experiment demonstrates how to test a model and how the results are generated and unfolded.

Test results of a model include:

1. Top-1 and top-5 accuracies on five SPASL components with the corresponding SPASL score calculated ('./demo/result/result_on_whole_SPASL/{model}/{model}_top1(and 5)_acc.csv').

2. An entry of the tested model performance added to the local SPASL test history ('./demo/result/result_on_whole_SPASL/history.csv(and .json)').

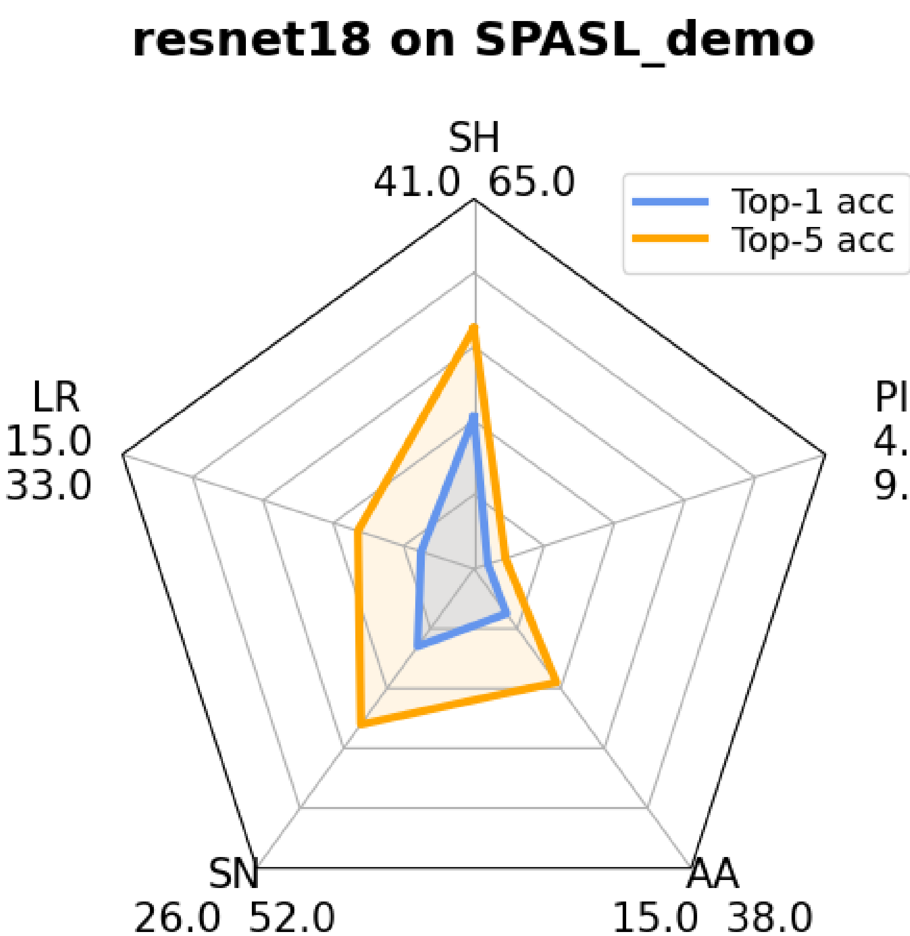
3. IW tables (Optional, provided in the experiment at './demo/result/IW_table/{model}').

4. Radar charts (Optional, provided in the experiment at './demo/result/IW_table/{model}').

When assessing a model using the authentic SPASL_v1, test results will be provided in the same folder hierarchy.

[28]:
```
# First, we test ResNet-18 on SPASL_demo.

model = models.resnet18(weights='DEFAULT')
model_name = 'resnet18'

# The location of SPASL_demo
SPASL_dir = './demo/SPASL_demo'
# Location to store all test results
result_mother_folder = './demo/result/'
# Test the current model for the first time
mod_test.test_on_whole_SPASL_benchmark(model, model_name, SPASL_dir,␣
 ↪result_mother_folder, bm_name = 'SPASL_demo', draw_radar_graph = True,␣
 ↪progress_bar = True)
```

```
100%|          | 100/100 [00:03<00:00, 25.76it/s]

resnet18 on SH: # R1 = 41, # Wrong = 35, total # = 100. Top-1: 41.0%, Top-5:
65.0%

100%|          | 100/100 [00:02<00:00, 36.67it/s]

resnet18 on PI: # R1 = 4, # Wrong = 91, total # = 100. Top-1: 4.0%, Top-5: 9.0%

100%|          | 100/100 [00:03<00:00, 27.43it/s]

resnet18 on AA: # R1 = 15, # Wrong = 62, total # = 100. Top-1: 15.0%, Top-5:
38.0%

100%|          | 100/100 [00:03<00:00, 26.08it/s]

resnet18 on SN: # R1 = 26, # Wrong = 48, total # = 100. Top-1: 26.0%, Top-5:
52.0%

100%|          | 100/100 [00:02<00:00, 34.83it/s]

resnet18 on LR: # R1 = 15, # Wrong = 67, total # = 100. Top-1: 15.0%, Top-5:
33.0%
**********
Model resnet18 SPASL score: 3.24 and 13.53
**********
IW-tables are saved to ./demo/result//IW_table/resnet18
Summary (accuracies, scores) is saved to
./demo/result//result_on_whole_SPASL/resnet18
Summary is also added to test history at
./demo/result//result_on_whole_SPASL/history.csv
```

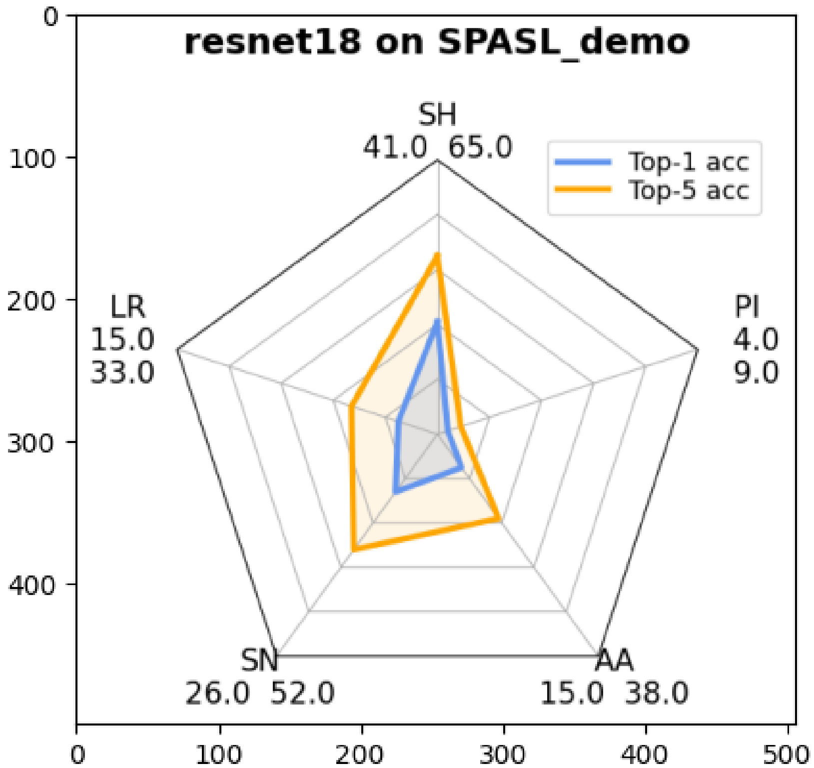



```
The radar chart is saved to ./demo/result//result_on_whole_SPASL/resnet18
```

[29]:

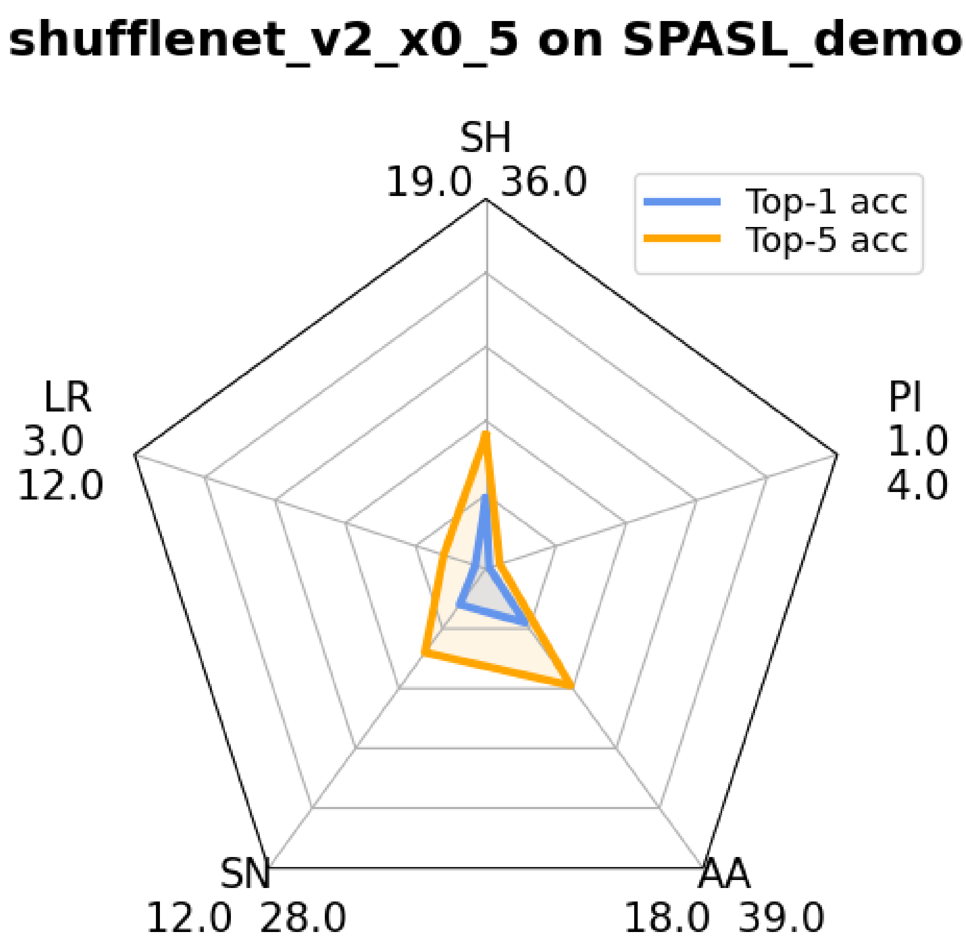
```
# When evaluating the same model in the future, test results will be directly␣
↪retrieved from the local test histroy.
mod_test.test_on_whole_SPASL_benchmark(model, model_name, SPASL_dir,␣
↪result_mother_folder, bm_name = 'SPASL_demo', draw_radar_graph = True,␣
↪progress_bar = True)
```

```
A record is found in the history:
     model    SH   PI    AA    SN    LR  score
0  resnet18  41.0  4.0  15.0  26.0  15.0   3.24
```



[30]:
```
# We encourage you to try as many models as you would like to.
# We provide several here.
# More models can be found at  "https://pytorch.org/vision/stable/models.html"

#**********************************************
#MaxVit
#model = models.maxvit_t(weights='DEFAULT')
#model_name = 'maxvit_t'

#ShuffleNet_v2_x0_5
model = models.shufflenet_v2_x0_5(weights='DEFAULT')
model_name = 'shufflenet_v2_x0_5'

#VitB-16 (330 MB, download may take longer)
#model = models.vit_b_16(weights='DEFAULT')
#model_name = 'vit_b_16'
#**********************************************

mod_test.test_on_whole_SPASL_benchmark(model, model_name, SPASL_dir,␣
 ↪result_mother_folder, bm_name = 'SPASL_demo', draw_radar_graph = True,␣
 ↪progress_bar = True)
```

```
100%|     | 100/100 [00:04<00:00, 24.45it/s]

shufflenet_v2_x0_5 on SH: # R1 = 19, # Wrong = 64, total # = 100. Top-1: 19.0%,
Top-5: 36.0%

100%|     | 100/100 [00:03<00:00, 32.47it/s]

shufflenet_v2_x0_5 on PI: # R1 = 1, # Wrong = 96, total # = 100. Top-1: 1.0%,
Top-5: 4.0%

100%|     | 100/100 [00:03<00:00, 26.71it/s]

shufflenet_v2_x0_5 on AA: # R1 = 18, # Wrong = 61, total # = 100. Top-1: 18.0%,
Top-5: 39.0%

100%|     | 100/100 [00:03<00:00, 27.47it/s]

shufflenet_v2_x0_5 on SN: # R1 = 12, # Wrong = 72, total # = 100. Top-1: 12.0%,
Top-5: 28.0%

100%|     | 100/100 [00:03<00:00, 31.41it/s]

shufflenet_v2_x0_5 on LR: # R1 = 3, # Wrong = 88, total # = 100. Top-1: 3.0%,
Top-5: 12.0%
**********
Model shufflenet_v2_x0_5 SPASL score: 0.69 and 4.32
**********
IW-tables are saved to ./demo/result//IW_table/shufflenet_v2_x0_5
Summary (accuracies, scores) is saved to
./demo/result//result_on_whole_SPASL/shufflenet_v2_x0_5
Summary is also added to test history at
./demo/result//result_on_whole_SPASL/history.csv
```




```
The radar chart is saved to
./demo/result//result_on_whole_SPASL/shufflenet_v2_x0_5
```

```
[ ]:
```

FAQ

Q1. How to solve the error "…/Activate.ps1 cannot be loaded because running scripts is disabled on this system." when trying to activate a virtual env?

A1. Enter the command "Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass", then activate the virtual env.