

```
import os
import pandas as pd
```

```
s=pd.read_csv("ham_metadata.csv")
s.head()
```

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

```
d="images for cancer"
from PIL import Image
import numpy as np
l=[]
m=[]
images={}
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    l.append(full_path)
    m.append(path)
```

```
d="images for cancer2"
for path in os.listdir(d):
    full_path=os.path.join(d,path)
    l.append(full_path)
    m.append(path)
l=np.array(l)
m=np.array(m)
```

```
for i in range(len(l)):
    images[m[i]]=np.asarray(Image.open(l[i]).resize((299,299)))
```

```
count=0
images1={}
for i in images:
    images1[i[:-4]]=images[i]
    count+=1
print(images[m[0]].shape)
print(len(images1))
```

```
(299, 299, 3)
10015
```

```
s1=[]
for i in range(100000):
```

```

for i in range(len(s)):
    img=s["image_id"][i]
    if(img in images1):
        s1.append(images1[img])
    else:
        s1.append("None")

```

```

s["image"]=s1
print(s1.count("None"))
s.head()
print(s.columns)
df=s[s.image!="None"]
df.head()

```

C:\Users\shiri\anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning

```

0
Index(['lesion_id', 'image_id', 'dx', 'dx_type', 'age', 'sex', 'localization',
      'image'],
      dtype='object')
C:\Users\shiri\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py:57: FutureWarning
result = libops.scalar_compare(x.ravel(), y, op)

```

	lesion_id	image_id	dx	dx_type	age	sex	localization	image
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	[[[187, 149, 191], [188, 151, 192], [190, 155, ...
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	[[[25, 13, 22], [25, 14, 22], [26, 13, 23], [2...
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	[[[186, 128, 136], [186, 127, 135], [190, 131, ...
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	[[[23, 10, 15], [24, 11, 16], [24, 11, 19], [2...
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	[[[123, 82, 104], [129, 86, 109], [133, 91, 11...

```

print(len(df))
k=[]
xtrain=df["image"]
print(len(xtrain))
print(xtrain[0].shape)

10015
10015
(299, 299, 3)

```

Start coding or [generate](#) with AI.

```
import tensorflow as tf
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
from keras.models import Sequential
```

Using TensorFlow backend.

```
model = Sequential()
model.add(Conv2D(16,(3,3),input_shape=(75,100,3),padding= "same",activation="relu"))
model.add(Conv2D(16,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3),activation="relu"))
model.add(Conv2D(32,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3),activation="relu"))
model.add(Flatten(input_shape=(75,100,3)))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.4))
model.add(Dense(128,activation="relu"))
model.add(Dense(62,activation="sigmoid"))
model.add(Dense(7,activation="softmax"))
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 75, 100, 16)	448
conv2d_2 (Conv2D)	(None, 73, 98, 16)	2320
max_pooling2d_1 (MaxPooling2	(None, 36, 49, 16)	0
conv2d_3 (Conv2D)	(None, 34, 47, 32)	4640
conv2d_4 (Conv2D)	(None, 32, 45, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 16, 22, 32)	0
conv2d_5 (Conv2D)	(None, 14, 20, 64)	18496
max_pooling2d_3 (MaxPooling2	(None, 7, 10, 64)	0
conv2d_6 (Conv2D)	(None, 5, 8, 64)	36928
flatten_1 (Flatten)	(None, 2560)	0
dense_1 (Dense)	(None, 128)	327808
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 128)	16512
dense_4 (Dense)	(None, 62)	7998

```
dense_5 (Dense)                (None, 7)                441
=====
Total params: 441,351
Trainable params: 441,351
Non-trainable params: 0
```

```
model.compile(loss="sparse_categorical_crossentropy",optimizer=tf.keras.optimizers.Adam(learning_r
,metrics=["accuracy"])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator,array_to_img,load_img,img_to_a
datagen=ImageDataGenerator(
    rescale=1/255)
```

```
hist=model.fit_generator(datagen.flow(xtrain,ytrain,batch_size=100),validation_data=(xtest,ytest),
```

```
Epoch 1/50
185/185 [=====] - 404s 2s/step - loss: 1.8795 - accuracy: 0.2124 - va
Epoch 2/50
185/185 [=====] - 200s 1s/step - loss: 1.6420 - accuracy: 0.3314 - va
Epoch 3/50
185/185 [=====] - 201s 1s/step - loss: 1.5402 - accuracy: 0.3738 - va
Epoch 4/50
185/185 [=====] - 200s 1s/step - loss: 1.4540 - accuracy: 0.4196 - va
Epoch 5/50
185/185 [=====] - 197s 1s/step - loss: 1.3976 - accuracy: 0.4471 - va
Epoch 6/50
185/185 [=====] - 206s 1s/step - loss: 1.3140 - accuracy: 0.4845 - va
Epoch 7/50
185/185 [=====] - 205s 1s/step - loss: 1.2619 - accuracy: 0.5082 - va
Epoch 8/50
185/185 [=====] - 208s 1s/step - loss: 1.2225 - accuracy: 0.5224 - va
Epoch 9/50
185/185 [=====] - 208s 1s/step - loss: 1.1824 - accuracy: 0.5436 - va
Epoch 10/50
185/185 [=====] - 207s 1s/step - loss: 1.1395 - accuracy: 0.5606 - va
Epoch 11/50
185/185 [=====] - 204s 1s/step - loss: 1.1488 - accuracy: 0.5591 - va
Epoch 12/50
185/185 [=====] - 206s 1s/step - loss: 1.0847 - accuracy: 0.5813 - va
Epoch 13/50
185/185 [=====] - 206s 1s/step - loss: 1.0543 - accuracy: 0.5953 - va
Epoch 14/50
185/185 [=====] - 207s 1s/step - loss: 1.0268 - accuracy: 0.6018 - va
Epoch 15/50
185/185 [=====] - 209s 1s/step - loss: 1.0074 - accuracy: 0.6108 - va
Epoch 16/50
185/185 [=====] - 207s 1s/step - loss: 0.9903 - accuracy: 0.6186 - va
Epoch 17/50
185/185 [=====] - 208s 1s/step - loss: 0.9516 - accuracy: 0.6318 - va
Epoch 18/50
185/185 [=====] - 209s 1s/step - loss: 0.9344 - accuracy: 0.6407 - va
Epoch 19/50
185/185 [=====] - 208s 1s/step - loss: 0.9257 - accuracy: 0.6448 - va
Epoch 20/50
185/185 [=====] - 208s 1s/step - loss: 0.9073 - accuracy: 0.6547 - va
Epoch 21/50
185/185 [=====] - 153s 828ms/step - loss: 0.8840 - accuracy: 0.6621 -
Epoch 22/50
185/185 [=====] - 164s 888ms/step - loss: 0.8617 - accuracy: 0.6695 -
Epoch 23/50
185/185 [=====] - 173s 933ms/step - loss: 0.8415 - accuracy: 0.6755 -
```

```

Epoch 24/50
185/185 [=====] - 182s 982ms/step - loss: 0.8345 - accuracy: 0.6780 -
Epoch 25/50
185/185 [=====] - 174s 940ms/step - loss: 0.8083 - accuracy: 0.6894 -
Epoch 26/50
185/185 [=====] - 169s 915ms/step - loss: 0.7981 - accuracy: 0.6933 -
Epoch 27/50
185/185 [=====] - 170s 917ms/step - loss: 0.7789 - accuracy: 0.7046 -
Epoch 28/50
185/185 [=====] - 174s 942ms/step - loss: 0.7672 - accuracy: 0.7030 -
Epoch 29/50
185/185 [=====] - 168s 908ms/step - loss: 0.7440 - accuracy: 0.7144 -

```

```
hist1=model.evaluate(xtest,ytest)
```

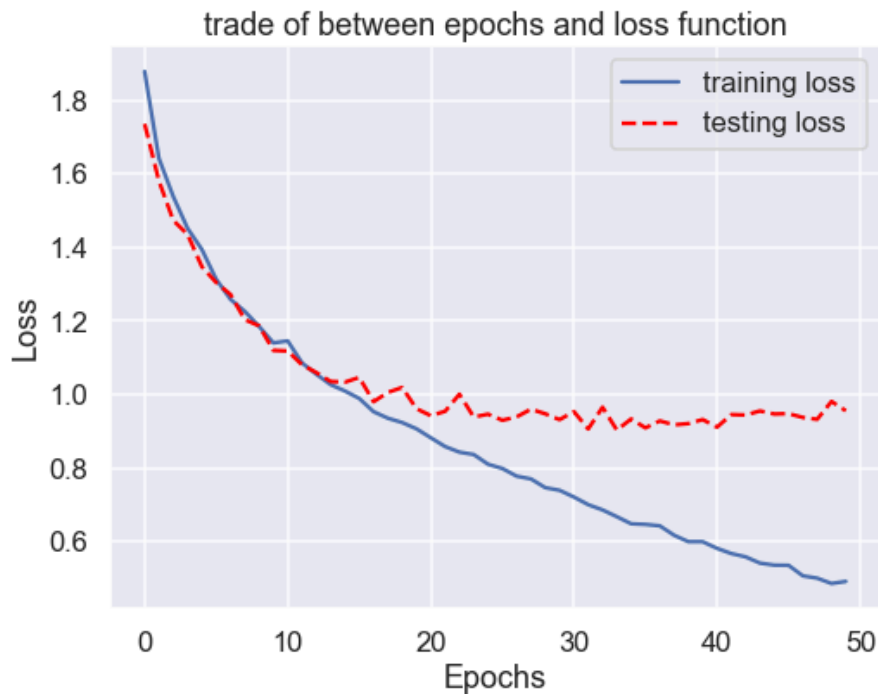
```
5287/5287 [=====] - 11s 2ms/step
```

```

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context('talk')
plt.figure(figsize=(8,6))
plt.title("trade of between epochs and loss function")
#plt.plot(range(0,5),hist.history["loss"])
plt.plot(range(0,50),hist.history["loss"],label="training loss")
plt.plot(range(0,50),hist.history["val_loss"],linestyle="--",label="testing loss",color="red")
plt.xlabel("Epochs")
plt.legend()
plt.ylabel("Loss")

```

```
Text(0, 0.5, 'Loss')
```



```

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_context('talk')
plt.figure(figsize=(8,6))
plt.title("trade of between epochs and accuracy")
#plt.plot(range(0,5),hist.history["accuracy"])
plt.plot(range(0,50),hist.history["val_accuracy"],linestyle="--",label="validation accuracy")
plt.plot(range(0,50),hist.history["accuracy"],label="training accuracy",color="red")
plt.xlabel("Epochs")
plt.legend()
plt.ylabel("Accuracy")

```

Text(0, 0.5, 'Accuracy')



```

d1= {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}

```

```

d2= {
    'nv':0,
    'mel':1,
    'bkl':2,
    'bcc':3,
    'akiec':4,
    'vasc':5,
    'df':6
}

```

```
label=[];type=[]
for i in range(len(df)):
    label.append(d2[df["dx"][i]])
    type.append(d1[df["dx"][i]])
df["label"]=label
df["type"]=type
```

```
df.head()
```

	lesion_id	image_id	dx	dx_type	age	sex	localization	ima
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	[[[190, 153, 194], [192, 154, 196], [191, 153, 195], [193, 155, 197], [194, 156, 198], [195, 157, 199], [196, 158, 200], [197, 159, 201], [198, 160, 202], [199, 161, 203], [200, 162, 204], [201, 163, 205], [202, 164, 206], [203, 165, 207], [204, 166, 208], [205, 167, 209], [206, 168, 210], [207, 169, 211], [208, 170, 212], [209, 171, 213], [210, 172, 214], [211, 173, 215], [212, 174, 216], [213, 175, 217], [214, 176, 218], [215, 177, 219], [216, 178, 220], [217, 179, 221], [218, 180, 222], [219, 181, 223], [220, 182, 224], [221, 183, 225], [222, 184, 226], [223, 185, 227], [224, 186, 228], [225, 187, 229], [226, 188, 230], [227, 189, 231], [228, 190, 232], [229, 191, 233], [230, 192, 234], [231, 193, 235], [232, 194, 236], [233, 195, 237], [234, 196, 238], [235, 197, 239], [236, 198, 240], [237, 199, 241], [238, 200, 242], [239, 201, 243], [240, 202, 244], [241, 203, 245], [242, 204, 246], [243, 205, 247], [244, 206, 248], [245, 207, 249], [246, 208, 250], [247, 209, 251], [248, 210, 252], [249, 211, 253], [250, 212, 254], [251, 213, 255], [252, 214, 256], [253, 215, 257], [254, 216, 258], [255, 217, 259], [256, 218, 260], [257, 219, 261], [258, 220, 262], [259, 221, 263], [260, 222, 264], [261, 223, 265], [262, 224, 266], [263, 225, 267], [264, 226, 268], [265, 227, 269], [266, 228, 270], [267, 229, 271], [268, 230, 272], [269, 231, 273], [270, 232, 274], [271, 233, 275], [272, 234, 276], [273, 235, 277], [274, 236, 278], [275, 237, 279], [276, 238, 280], [277, 239, 281], [278, 240, 282], [279, 241, 283], [280, 242, 284], [281, 243, 285], [282, 244, 286], [283, 245, 287], [284, 246, 288], [285, 247, 289], [286, 248, 290], [287, 249, 291], [288, 250, 292], [289, 251, 293], [290, 252, 294], [291, 253, 295], [292, 254, 296], [293, 255, 297], [294, 256, 298], [295, 257, 299], [296, 258, 300], [297, 259, 301], [298, 260, 302], [299, 261, 303], [300, 262, 304], [301, 263, 305], [302, 264, 306], [303, 265, 307], [304, 266, 308], [305, 267, 309], [306, 268, 310], [307, 269, 311], [308, 270, 312], [309, 271, 313], [310, 272, 314], [311, 273, 315], [312, 274, 316], [313, 275, 317], [314, 276, 318], [315, 277, 319], [316, 278, 320], [317, 279, 321], [318, 280, 322], [319, 281, 323], [320, 282, 324], [321, 283, 325], [322, 284, 326], [323, 285, 327], [324, 286, 328], [325, 287, 329], [326, 288, 330], [327, 289, 331], [328, 290, 332], [329, 291, 333], [330, 292, 334], [331, 293, 335], [332, 294, 336], [333, 295, 337], [334, 296, 338], [335, 297, 339], [336, 298, 340], [337, 299, 341], [338, 300, 342], [339, 301, 343], [340, 302, 344], [341, 303, 345], [342, 304, 346], [343, 305, 347], [344, 306, 348], [345, 307, 349], [346, 308, 350], [347, 309, 351], [348, 310, 352], [349, 311, 353], [350, 312, 354], [351, 313, 355], [352, 314, 356], [353, 315, 357], [354, 316, 358], [355, 317, 359], [356, 318, 360], [357, 319, 361], [358, 320, 362], [359, 321, 363], [360, 322, 364], [361, 323, 365], [362, 324, 366], [363, 325, 367], [364, 326, 368], [365, 327, 369], [366, 328, 370], [367, 329, 371], [368, 330, 372], [369, 331, 373], [370, 332, 374], [371, 333, 375], [372, 334, 376], [373, 335, 377], [374, 336, 378], [375, 337, 379], [376, 338, 380], [377, 339, 381], [378, 340, 382], [379, 341, 383], [380, 342, 384], [381, 343, 385], [382, 344, 386], [383, 345, 387], [384, 346, 388], [385, 347, 389], [386, 348, 390], [387, 349, 391], [388, 350, 392], [389, 351, 393], [390, 352, 394], [391, 353, 395], [392, 354, 396], [393, 355, 397], [394, 356, 398], [395, 357, 399], [396, 358, 400], [397, 359, 401], [398, 360, 402], [399, 361, 403], [400, 362, 404], [401, 363, 405], [402, 364, 406], [403, 365, 407], [404, 366, 408], [405, 367, 409], [406, 368, 410], [407, 369, 411], [408, 370, 412], [409, 371, 413], [410, 372, 414], [411, 373, 415], [412, 374, 416], [413, 375, 417], [414, 376, 418], [415, 377, 419], [416, 378, 420], [417, 379, 421], [418, 380, 422], [419, 381, 423], [420, 382, 424], [421, 383, 425], [422, 384, 426], [423, 385, 427], [424, 386, 428], [425, 387, 429], [426, 388, 430], [427, 389, 431], [428, 390, 432], [429, 391, 433], [430, 392, 434], [431, 393, 435], [432, 394, 436], [433, 395, 437], [434, 396, 438], [435, 397, 439], [436, 398, 440], [437, 399, 441], [438, 400, 442], [439, 401, 443], [440, 402, 444], [441, 403, 445], [442, 404, 446], [443, 405, 447], [444, 406, 448], [445, 407, 449], [446, 408, 450], [447, 409, 451], [448, 410, 452], [449, 411, 453], [450, 412, 454], [451, 413, 455], [452, 414, 456], [453, 415, 457], [454, 416, 458], [455, 417, 459], [456, 418, 460], [457, 419, 461], [458, 420, 462], [459, 421, 463], [460, 422, 464], [461, 423, 465], [462, 424, 466], [463, 425, 467], [464, 426, 468], [465, 427, 469], [466, 428, 470], [467, 429, 471], [468, 430, 472], [469, 431, 473], [470, 432, 474], [471, 433, 475], [472, 434, 476], [473, 435, 477], [474, 436, 478], [475, 437, 479], [476, 438, 480], [477, 439, 481], [478, 440, 482], [479, 441, 483], [480, 442, 484], [481, 443, 485], [482, 444, 486], [483, 445, 487], [484, 446, 488], [485, 447, 489], [486, 448, 490], [487, 449, 491], [488, 450, 492], [489, 451, 493], [490, 452, 494], [491, 453, 495], [492, 454, 496], [493, 455, 497], [494, 456, 498], [495, 457, 499], [496, 458, 500], [497, 459, 501], [498, 460, 502], [499, 461, 503], [500, 462, 504], [501, 463, 505], [502, 464, 506], [503, 465, 507], [504, 466, 508], [505, 467, 509], [506, 468, 510], [507, 469, 511], [508, 470, 512], [509, 471, 513], [510, 472, 514], [511, 473, 515], [512, 474, 516], [513, 475, 517], [514, 476, 518], [515, 477, 519], [516, 478, 520], [517, 479, 521], [518, 480, 522], [519, 481, 523], [520, 482, 524], [521, 483, 525], [522, 484, 526], [523, 485, 527], [524, 486, 528], [525, 487, 529], [526, 488, 530], [527, 489, 531], [528, 490, 532], [529, 491, 533], [530, 492, 534], [531, 493, 535], [532, 494, 536], [533, 495, 537], [534, 496, 538], [535, 497, 539], [536, 498, 540], [537, 499, 541], [538, 500, 542], [539, 501, 543], [540, 502, 544], [541, 503, 545], [542, 504, 546], [543, 505, 547], [544, 506, 548], [545, 507, 549], [546, 508, 550], [547, 509, 551], [548, 510, 552], [549, 511, 553], [550, 512, 554], [551, 513, 555], [552, 514, 556], [553, 515, 557], [554, 516, 558], [555, 517, 559], [556, 518, 560], [557, 519, 561], [558, 520, 562], [559, 521, 563], [560, 522, 564], [561, 523, 565], [562, 524, 566], [563, 525, 567], [564, 526, 568], [565, 527, 569], [566, 528, 570], [567, 529, 571], [568, 530, 572], [569, 531, 573], [570, 532, 574], [571, 533, 575], [572, 534, 576], [573, 535, 577], [574, 536, 578], [575, 537, 579], [576, 538, 580], [577, 539, 581], [578, 540, 582], [579, 541, 583], [580, 542, 584], [581, 543, 585], [582, 544, 586], [583, 545, 587], [584, 546, 588], [585, 547, 589], [586, 548, 590], [587, 549, 591], [588, 550, 592], [589, 551, 593], [590, 552, 594], [591, 553, 595], [592, 554, 596], [593, 555, 597], [594, 556, 598], [595, 557, 599], [596, 558, 600], [597, 559, 601], [598, 560, 602], [599, 561, 603], [600, 562, 604], [601, 563, 605], [602, 564, 606], [603, 565, 607], [604, 566, 608], [605, 567, 609], [606, 568, 610], [607, 569, 611], [608, 570, 612], [609, 571, 613], [610, 572, 614], [611, 573, 615], [612, 574, 616], [613, 575, 617], [614, 576, 618], [615, 577, 619], [616, 578, 620], [617, 579, 621], [618, 580, 622], [619, 581, 623], [620, 582, 624], [621, 583, 625], [622, 584, 626], [623, 585, 627], [624, 586, 628], [625, 587, 629], [626, 588, 630], [627, 589, 631], [628, 590, 632], [629, 591, 633], [630, 592, 634], [631, 593, 635], [632, 594, 636], [633, 595, 637], [634, 596, 638], [635, 597, 639], [636, 598, 640], [637, 599, 641], [638, 600, 642], [639, 601, 643], [640, 602, 644], [641, 603, 645], [642, 604, 646], [643, 605, 647], [644, 606, 648], [645, 607, 649], [646, 608, 650], [647, 609, 651], [648, 610, 652], [649, 611, 653], [650, 612, 654], [651, 613, 655], [652, 614, 656], [653, 615, 657], [654, 616, 658], [655, 617, 659], [656, 618, 660], [657, 619, 661], [658, 620, 662], [659, 621, 663], [660, 622, 664], [661, 623, 665], [662, 624, 666], [663, 625, 667], [664, 626, 668], [665, 627, 669], [666, 628, 670], [667, 629, 671], [668, 630, 672], [669, 631, 673], [670, 632, 674], [671, 633, 675], [672, 634, 676], [673, 635, 677], [674, 636, 678], [675, 637, 679], [676, 638, 680], [677, 639, 681], [678, 640, 682], [679, 641, 683], [680, 642, 684], [681, 643, 685], [682, 644, 686], [683, 645, 687], [684, 646, 688], [685, 647, 689], [686, 648, 690], [687, 649, 691], [688, 650, 692], [689, 651, 693], [690, 652, 694], [691, 653, 695], [692, 654, 696], [693, 655, 697], [694, 656, 698], [695, 657, 699], [696, 658, 700], [697, 659, 701], [698, 660, 702], [699, 661, 703], [700, 662, 704], [701, 663, 705], [702, 664, 706], [703, 665, 707], [704, 666, 708], [705, 667, 709], [706, 668, 710], [707, 669, 711], [708, 670, 712], [709, 671, 713], [710, 672, 714], [711, 673, 715], [712, 674, 716], [713, 675, 717], [714, 676, 718], [715, 677, 719], [716, 678, 720], [717, 679, 721], [718, 680, 722], [719, 681, 723], [720, 682, 724], [721, 683, 725], [722, 684, 726], [723, 685, 727], [724, 686, 728], [725, 687, 729], [726, 688, 730], [727, 689, 731], [728, 690, 732], [729, 691, 733], [730, 692, 734], [731, 693, 735], [732, 694, 736], [733, 695, 737], [734, 696, 738], [735, 697, 739], [736, 698, 740], [737, 699, 741], [738, 700, 742], [739, 701, 743], [740, 702, 744], [741, 703, 745], [742, 704, 746], [743, 705, 747], [744, 706, 748], [745, 707, 749], [746, 708, 750], [747, 709, 751], [748, 710, 752], [749, 711, 753], [750, 712, 754], [751, 713, 755], [752, 714, 756], [753, 715, 757], [754, 716, 758], [755, 717, 759], [756, 718, 760], [757, 719, 761], [758, 720, 762], [759, 721, 763], [760, 722, 764], [761, 723, 765], [762, 724, 766], [763, 725, 767], [764, 726, 768], [765, 727, 769], [766, 728, 770], [767, 729, 771], [768, 730, 772], [769, 731, 773], [770, 732, 774], [771, 733, 775], [772, 734, 776], [773, 735, 777], [774, 736, 778], [775, 737, 779], [776, 738, 780], [777, 739, 781], [778, 740, 782], [779, 741, 783], [780, 742, 784], [781, 743, 785], [782, 744, 786], [783, 745, 787], [784, 746, 788], [785, 747, 789], [786, 748, 790], [787, 749, 791], [788, 750, 792], [789, 751, 793], [790, 752, 794], [791, 753, 795], [792, 754, 796], [793, 755, 797], [794, 756, 798], [795, 757, 799], [796, 758, 800], [797, 759, 801], [798, 760, 802], [799, 761, 803], [800, 762, 804], [801, 763, 805], [802, 764, 806], [803, 765, 807], [804, 766, 808], [805, 767, 809], [806, 768, 810], [807, 769, 811], [808, 770, 812], [809, 771, 813], [810, 772, 814], [811, 773, 815], [812, 774, 816], [813, 775, 817], [814, 776, 818], [815, 777, 819], [816, 778, 820], [817, 779, 821], [818, 780, 822], [819, 781, 823], [820, 782, 824], [821, 783, 825], [822, 784, 826], [823, 785, 827], [824, 786, 828], [825, 787, 829], [826, 788, 830], [827, 789, 831], [828, 790, 832], [829, 791, 833], [830, 792, 834], [831, 793, 835], [832, 794, 836], [833, 795, 837], [834, 796, 838], [835, 797, 839], [836, 798, 840], [837, 799, 841], [838, 800, 842], [839, 801, 843], [840, 802, 844], [841, 803, 845], [842, 804, 846], [843, 805, 847], [844, 806, 848], [845, 807, 849], [846, 808, 850], [847, 809, 851], [848, 810, 852], [849, 811, 853], [850, 812, 854], [851, 813, 855], [852, 814, 856], [853, 815, 857], [854, 816, 858], [855, 817, 859], [856, 818, 860], [857, 819, 861], [858, 820, 862], [859, 821, 863], [860, 822, 864], [861, 823, 865], [862, 824, 866], [863, 825, 867], [864, 826, 868], [865, 827, 869], [866, 828, 870], [867, 829, 871], [868, 830, 872], [869, 831, 873], [870, 832, 874], [871, 833, 875], [872, 834, 876], [873, 835, 877], [874, 836, 878], [875, 837, 879], [876, 838, 880], [877, 839, 881], [878, 840, 882], [879, 841, 883], [880, 842, 884], [881, 843, 885], [882, 844, 886], [883, 845, 887], [884, 846, 888], [885, 847, 889], [886, 848, 890], [887, 849, 891], [888, 850, 892], [889, 851, 893], [890, 852, 894], [891, 853, 895], [892, 854, 896], [893, 855, 897], [894, 856, 898], [895, 857, 899], [896, 858, 900], [897, 859, 901], [898, 860, 902], [899, 861, 903], [900, 862, 904], [901, 863, 905], [902, 864, 906], [903, 865, 907], [904, 866, 908], [905, 867, 909], [906, 868, 910], [907, 869, 911], [908, 870, 912], [909, 871, 913], [910, 872, 914], [911, 873, 915], [912, 874, 916], [913, 875, 917], [914, 876, 918], [915, 877, 919], [916, 878, 920], [917, 879, 921], [918, 880, 922], [919, 881, 923], [920, 882, 924], [921, 883, 925], [922, 884, 926], [923, 885, 927], [924, 886, 928], [925, 887, 929], [926, 888, 930], [927, 889, 931], [928, 890, 932], [929, 891, 933], [930, 892, 934], [931, 893, 935], [932, 894, 936], [933, 895, 937], [934, 896, 938], [935, 897, 939], [936, 898, 940], [937, 899, 941], [938, 900, 942], [939, 901, 943], [940, 902, 944], [941, 903, 945], [942, 904, 946], [943, 905, 947], [944, 906, 948], [945, 907, 949], [946, 908, 950], [947, 909, 951], [948, 910, 952], [949, 911, 953], [950, 912, 954], [951, 913, 955], [952, 914, 956], [953, 915, 957], [954, 916, 958], [955, 917, 959], [956, 918, 960], [957, 919, 961], [958, 920, 962], [959, 921, 963], [960, 922, 964], [961, 923, 965], [962, 924, 966], [963, 925, 967], [964, 926, 968], [965, 927, 969], [966, 928, 970], [967, 929, 971], [968, 930, 972], [969, 931, 973], [970, 932, 974], [971, 933, 975], [972, 934, 976], [973, 935, 977], [974, 936, 978], [975, 937, 979], [976, 938, 980], [977, 939, 981], [978, 940, 982], [979, 941, 983], [980, 942, 984], [981, 943, 985], [982, 944, 986], [983, 945, 987], [984, 946, 988], [985, 947, 989], [986, 948, 990], [987, 949, 991], [988, 950, 992], [989, 951, 993], [990, 952, 994], [991, 953, 995], [992, 954, 996], [993, 955, 997], [994, 956, 998], [995, 957, 999], [996, 958, 1

```

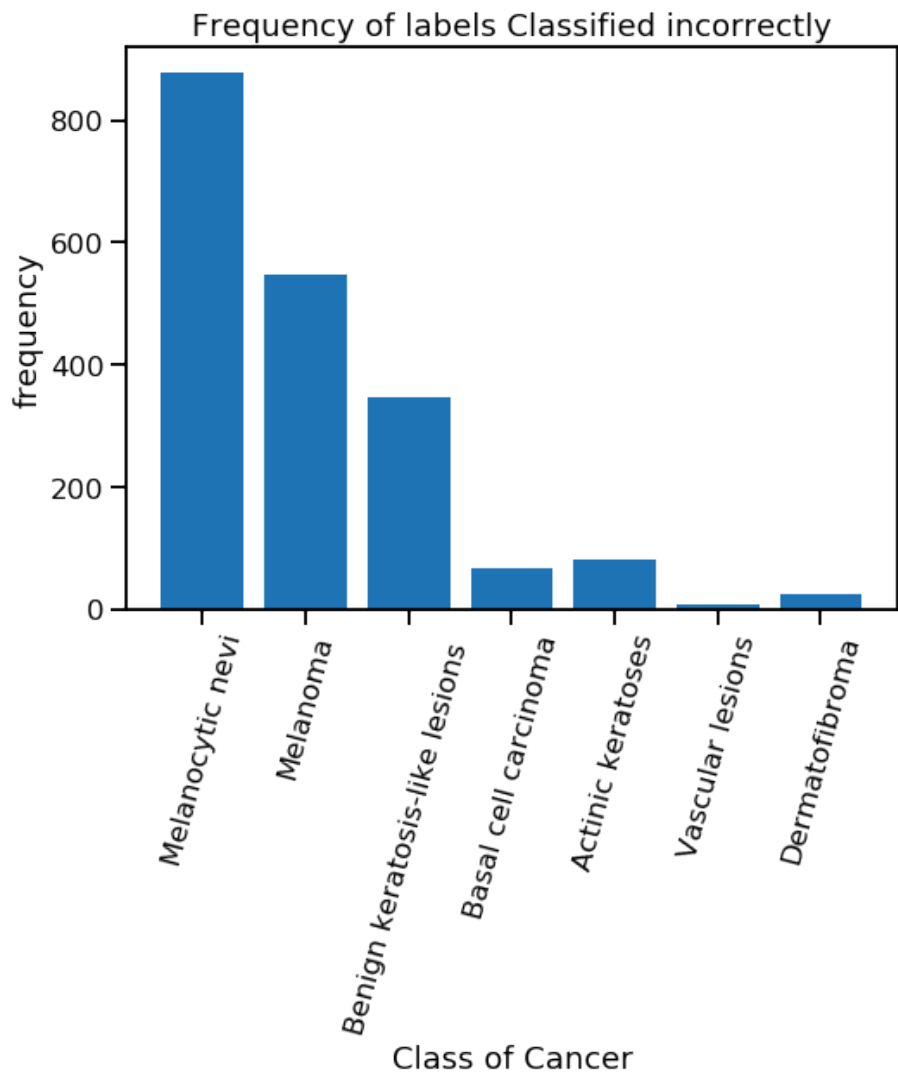
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(label,ypred)
acc=0
for i in range(len(ypred)):
    acc+=(ypred[i]==label[i])
print(acc)
print(cm)
cma=cm
cm=pd.DataFrame(cm)
emp=[]
emp2=[]
actual=[]
cm.index=x.index
cm.columns=x.index
for i in range(len(cma)):
    empc=0
    empw=0
    for j in range(len(cma)):
        if(i!=j):
            empc+=cma[i][j]
        else:
            empw+=cma[i][j]
    emp.append(empc)
    emp2.append(empw)
    actual.append(sum(cma[i]))
print(actual)

print(emp)
print(emp2)
plt.figure(figsize=(8,6))
plt.title("Frequency of labels Classified incorrectly")
plt.xlabel("Class of Cancer")
plt.ylabel("frequency")
plt.bar(x.index,emp)
plt.xticks(rotation="75")

```



```
8063
[[5828, 214, 415, 156, 38, 34, 20]
 [ 284, 566, 214, 22, 24, 1, 2]
 [ 226, 58, 753, 39, 20, 1, 2]
 [ 17, 1, 19, 446, 28, 2, 1]
 [ 0, 6, 33, 42, 246, 0, 0]
 [ 3, 1, 1, 1, 0, 134, 2]
 [ 13, 0, 5, 5, 2, 0, 90]]
[6705, 1113, 1099, 514, 327, 142, 115]
[877, 547, 346, 68, 81, 8, 25]
[5828, 566, 753, 446, 246, 134, 90]
([0, 1, 2, 3, 4, 5, 6], <a list of 7 Text xticklabel objects>)
```

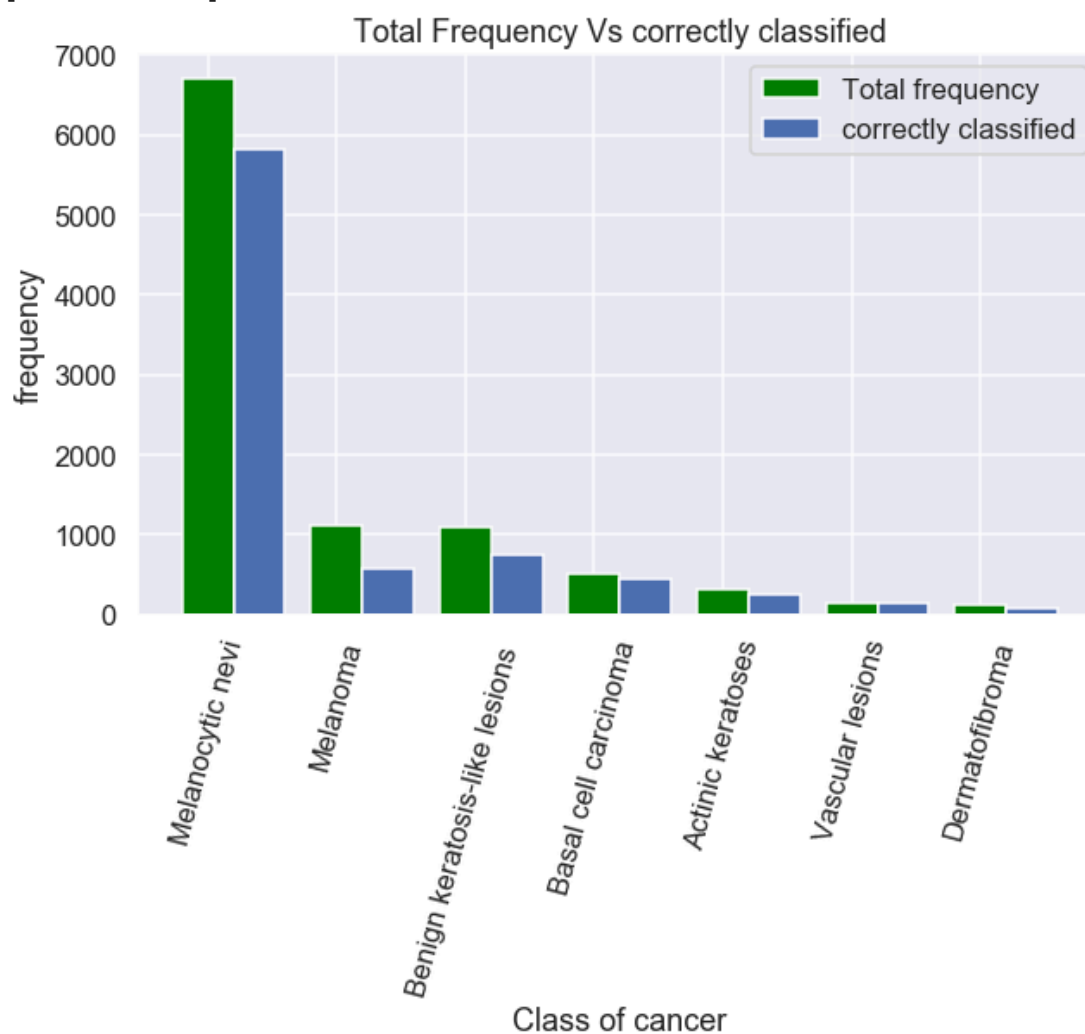


```

plt.figure(figsize=(10,6))
a=np.arange(7)
plt.title("Total Frequency Vs correctly classified")
plt.xlabel("Class of cancer")
plt.ylabel("frequency")
bar_width=0.4
print(a)
plt.bar(x.index,actual,width=bar_width,label="Total frequency",color="green")
plt.xticks(rotation="75")
plt.bar(a+bar_width,emp2,width=bar_width,label="correctly classified")
plt.legend()
plt.show()

```

[0 1 2 3 4 5 6]

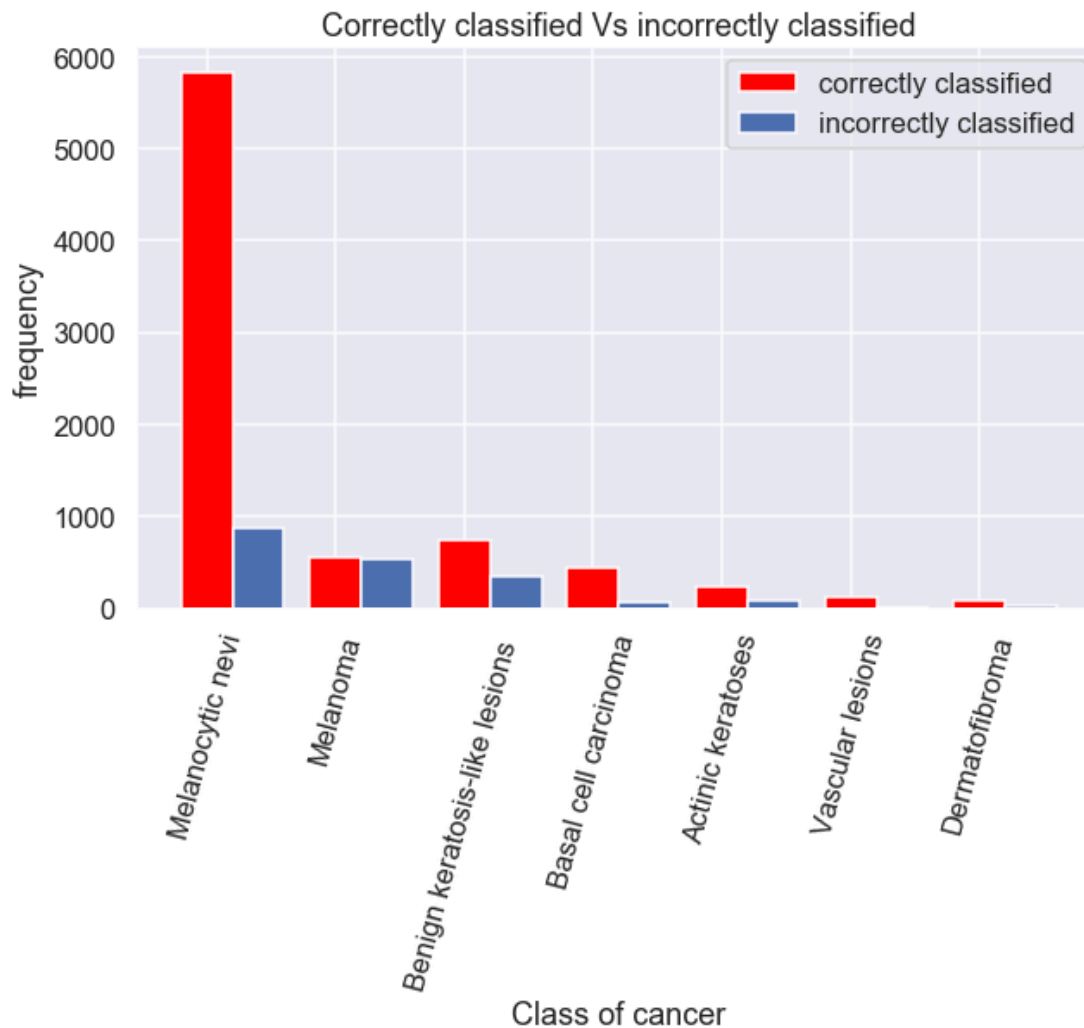


```

plt.figure(figsize=(10,6))
a=np.arange(7)
plt.title("Correctly classified Vs incorrectly classified")
plt.xlabel("Class of cancer")
plt.ylabel("frequency")
bar_width=0.4
print(a)
plt.bar(x.index,emp2,width=bar_width,label="correctly classified",color="red")
plt.xticks(rotation="75")
plt.bar(a+bar_width,emp,width=bar_width,label="incorrectly classified")
plt.legend()
plt.show()
lb=x.index

```

[0 1 2 3 4 5 6]



```

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(ytest,ypred1)
acc=0
print(cm)
cm=pd.DataFrame(cm)
cm.index=x.index
cm.columns=x.index

```

```

[[623  24  41  16   6   5   4]
 [ 99 429  90  38  62   5  15]
 [ 80 121 349  48  89   6  18]

```

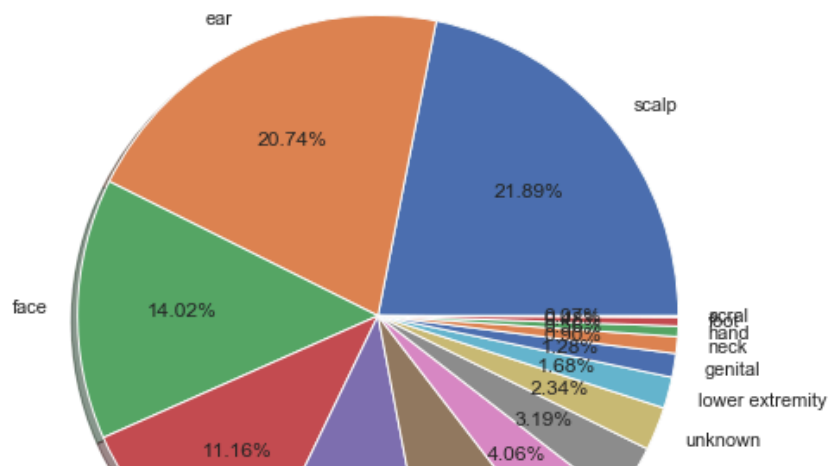
```
[ 13  22  39 401 111  22  31]
[   1  51  67 148 378   0  28]
[   0   2   1  10   2 512   3]
[   7  12  15  41  32   5 480]]
```

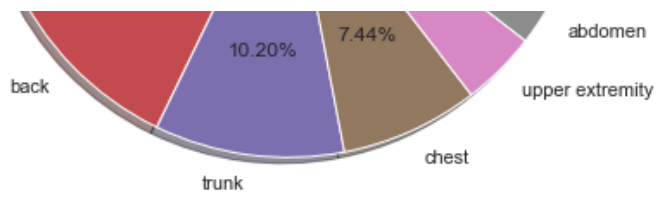
```
import matplotlib.pyplot as plt
vals=df["localization"].unique()
print(vals)
cnts=df["localization"].value_counts()
plt.figure(figsize=(8,8))
plt.pie(cnts,labels=vals,autopct="%1.2f%%",shadow=True,pctdistance=0.65)
```

```

['scalp' 'ear' 'face' 'back' 'trunk' 'chest' 'upper extremity' 'abdomen'
'unknown' 'lower extremity' 'genital' 'neck' 'hand' 'foot' 'acral']
([<matplotlib.patches.Wedge at 0x2d0dc77b708>,
<matplotlib.patches.Wedge at 0x2d0dc783208>,
<matplotlib.patches.Wedge at 0x2d0dc783e88>,
<matplotlib.patches.Wedge at 0x2d0dc78cbc8>,
<matplotlib.patches.Wedge at 0x2d0dc797b48>,
<matplotlib.patches.Wedge at 0x2d0dc7a1848>,
<matplotlib.patches.Wedge at 0x2d0dc7aa288>,
<matplotlib.patches.Wedge at 0x2d0dc7aaf08>,
<matplotlib.patches.Wedge at 0x2d0dc797888>,
<matplotlib.patches.Wedge at 0x2d0dc7a1588>,
<matplotlib.patches.Wedge at 0x2d0dc744e48>,
<matplotlib.patches.Wedge at 0x2d0dc7cf0c8>,
<matplotlib.patches.Wedge at 0x2d0dc7cfd48>,
<matplotlib.patches.Wedge at 0x2d0dc7d8a08>,
<matplotlib.patches.Wedge at 0x2d0dc7e26c8>],
[Text(0.850044637801859, 0.6981576567970206, 'scalp'),
Text(-0.48434367590701677, 0.9876290819983374, 'ear'),
Text(-1.0997116076210351, 0.02518690261938371, 'face'),
Text(-0.7910606058887917, -0.7643448945409774, 'back'),
Text(-0.14399207613610546, -1.0905348605202927, 'trunk'),
Text(0.4515570372814075, -1.0030434896262663, 'chest'),
Text(0.7770241418489049, -0.7786099684591594, 'upper extremity'),
Text(0.9327513585770874, -0.5830736686496809, 'abdomen'),
Text(1.0193916346283005, -0.413328797992399, 'unknown'),
Text(1.0632811366512243, -0.2818390044718449, 'lower extremity'),
Text(1.0848325959235923, -0.1820391134390633, 'genital'),
Text(1.0947359222518052, -0.10748609459595235, 'neck'),
Text(1.0985090016252645, -0.057253588082013494, 'hand'),
Text(1.0997919697781062, -0.021392129664751052, 'foot'),
Text(1.099997348309977, -0.0024153076448207023, 'acral')],
[Text(0.5022991041556439, 0.4125477062891485, '21.89%'),
Text(-0.28620308121778265, 0.5835990029990176, '20.74%'),
Text(-0.6498295863215208, 0.014883169729635829, '14.02%'),
Text(-0.46744490347974055, -0.4516583467742139, '11.16%'),
Text(-0.08508622680769867, -0.6444069630347184, '10.20%'),
Text(0.266829158393559, -0.5927075165973391, '7.44%'),
Text(0.45915062927435285, -0.46008770863495785, '4.06%'),
Text(0.5511712573410061, -0.3445435314748114, '3.19%'),
Text(0.6023677840985412, -0.24423974426823578, '2.34%'),
Text(0.6283024898393598, -0.16654122991518108, '1.68%'),
Text(0.641037443045759, -0.10756856703217375, '1.28%'),
Text(0.6468894086033393, -0.06351451044306275, '0.90%'),
Text(0.6491189555058381, -0.03383166568482615, '0.56%'),
Text(0.6498770730506991, -0.01264080389280744, '0.48%'),
Text(0.6499984330922591, -0.0014272272446667787, '0.07%')])

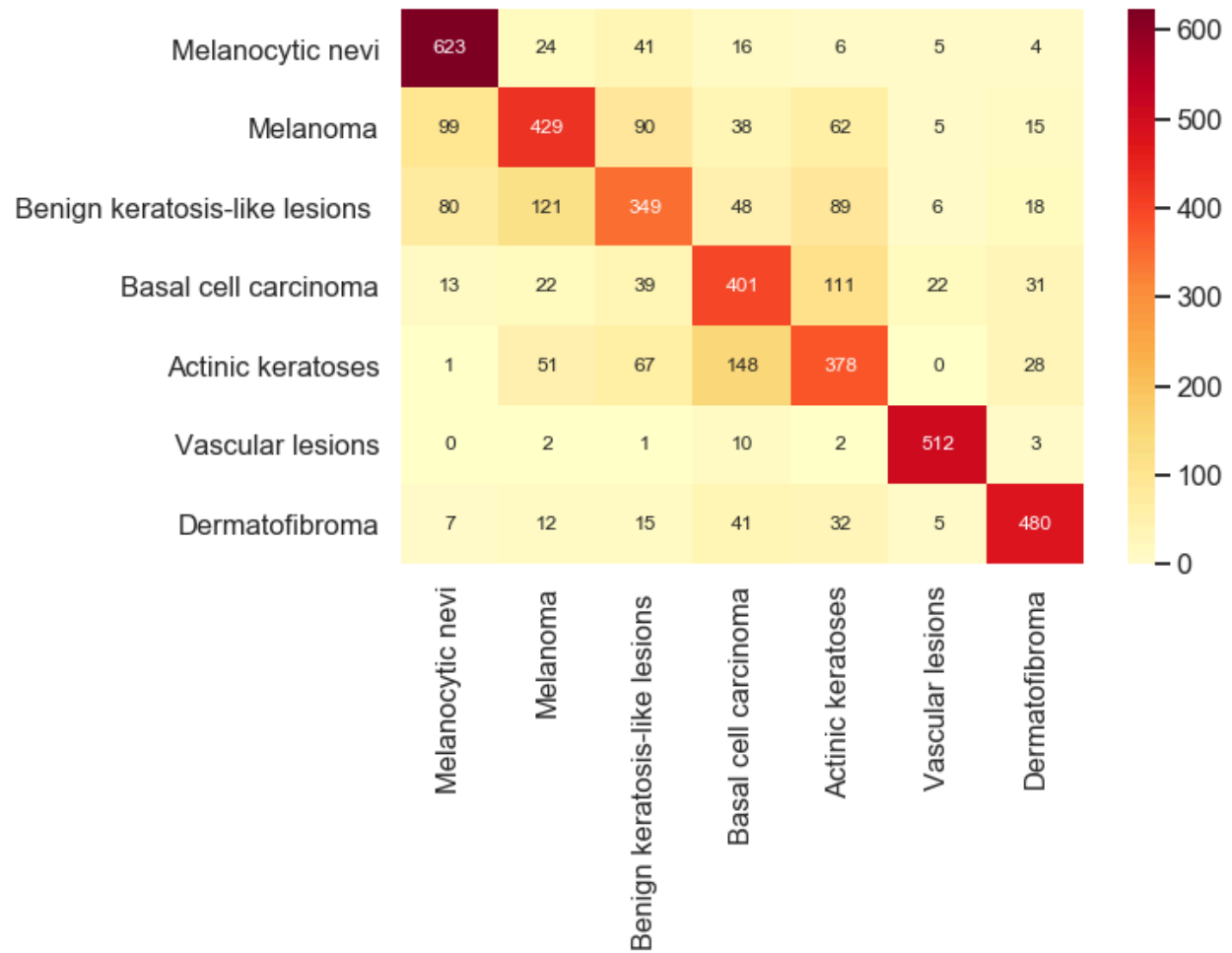
```





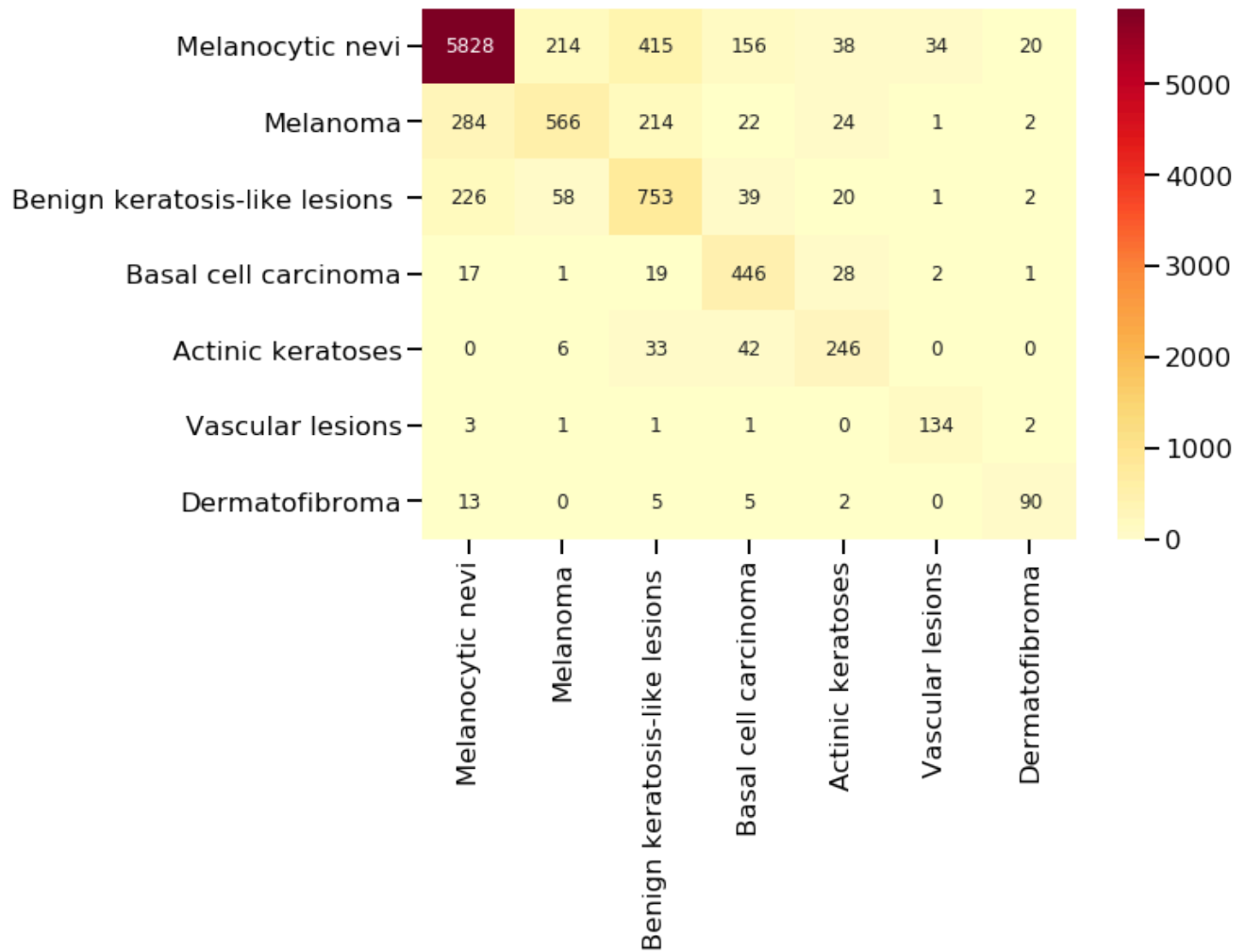

```
import seaborn as sn
plt.figure(figsize=(9,6))
sn.heatmap(cm, annot=True, annot_kws={"size": 12}, cmap= 'YlOrRd', fmt="g")
```


<matplotlib.axes._subplots.AxesSubplot at 0x279d27c9b08>



```
import seaborn as sn
plt.figure(figsize=(9,6))
sn.heatmap(cm, annot=True, annot_kws={"size": 12}, cmap= 'YlOrRd', fmt="g")
```

<matplotlib.axes._subplots.AxesSubplot at 0x279d4346c48>



```
ypred1=[]
for i in range(len(xtest)):
    ypred1.append(np.argmax(model.predict(xtest[i].reshape(1,75,100,3))))

#print(np.argmax(model.predict(xtest[1].reshape(1,75,100,3))))

s1=np.array(s1)

from sklearn import metrics
f_sco=metrics.precision_recall_fscore_support(label,ypred,average="weighted")
print(f_sco)0.8186782169714542

(0.8186782169714542, 0.8050923614578133, 0.807932103655772, None)

from sklearn import metrics
f_sco=metrics.precision_recall_fscore_support(ytest,ypred1,average="weighted")
print(f_sco)

(0.6854138384138033, 0.6892655367231638, 0.6856373865227726, None)
```

```

ypred=[]
for i in range(len(s1)):
    ypred.append(np.argmax(model.predict(s1[i].reshape(1,75,100,3))))
    if(i%1000==0):
        print(i)

```

```

0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

```

```
s1=np.array(s1)
```

```
s1=s1/255
```

```

s1=np.array(s1)
sdup=s1
ldup=label
print(s1.shape)

```

```
(10015, 75, 100, 3)
```

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(inp,oup,train_size=0.8)
```

```
xtest=xtest/255
```

```

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

```

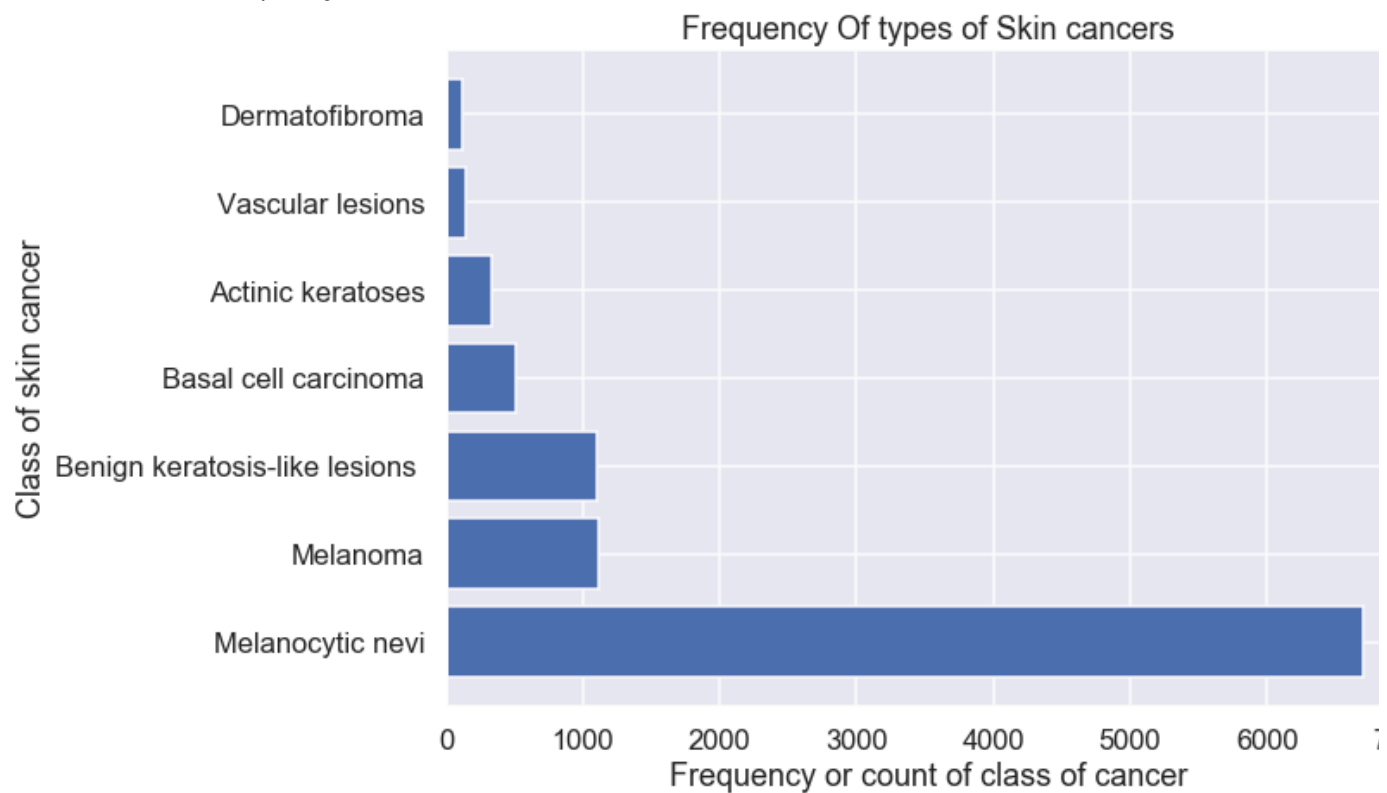
```

lbl=df["type"]
bars=["blue","red","green","yellow","indigo","pink","grey"]
x=pd.DataFrame(lbl.value_counts())
print(x)
import seaborn as sns
sns.set_context('talk')
ax=plt.figure(figsize=(10,7))
plt.title("Frequency Of types of Skin cancers")
plt.barh(x.index,x["type"])
plt.ylabel("Class of skin cancer")
#plt.legend(b,list(x.index))
plt.xlabel("Frequency or count of class of cancer")

```

	type
Melanocytic nevi	6705
Melanoma	1113
Benign keratosis-like lesions	1099
Basal cell carcinoma	514
Actinic keratoses	327
Vascular lesions	142
Dermatofibroma	115

Text(0.5, 0, 'Frequency or count of class of cancer')



```

li=[]
for i in range(len(x.index)):
    te=df[df.type==x.index[i]]
    valu=te["sex"].value_counts()
    valu=list(valu)
    if(len(valu)==2):
        valu.append(0)
    li.append(valu)
print(li)

```

```
[[3421, 3237, 47], [689, 424, 0], [626, 463, 10], [317, 197, 0], [221, 106, 0], [73, 69, 0], [0, 0, 0]]
```

```
male=[];female=[];unknown=[]
for i in range(len(li)):
    male.append(li[i][0])
    female.append(li[i][1])
    unknown.append(li[i][2])
```

```
print(male)
print(female)
```

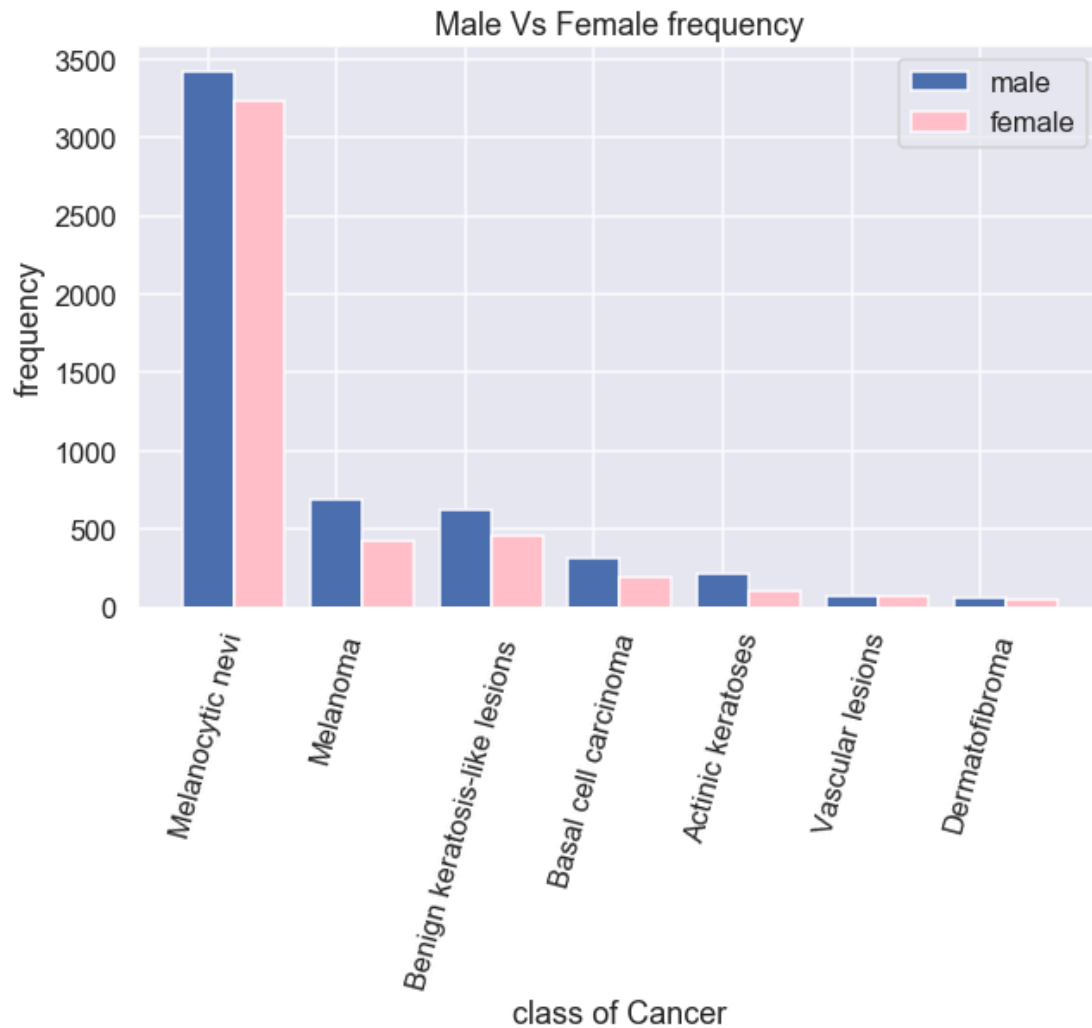
```
[3421, 689, 626, 317, 221, 73, 63]
[3237, 424, 463, 197, 106, 69, 52]
```

```
plt.figure(figsize=(10,6))
plt.title("Male Vs Female frequency")
plt.xlabel("class of Cancer")
plt.ylabel("frequency")
a=np.arange(7)
bar_width=0.4
print(a)

plt.bar(x.index,male,width=bar_width,label="male")
plt.xticks(rotation="75")
plt.bar(a+bar_width,female,width=bar_width,label="female",color="pink")
plt.legend()
plt.show()
```

```
lb=x.index
```

[0 1 2 3 4 5 6]



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator,array_to_img,load_img,img_to_a
datagen1=ImageDataGenerator(
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    brightness_range=None,
    shear_range=0.1,
    zoom_range=0.1,
)
```

```
derimages=[]
for i in range(len(df)):
    if(df["type"][i]=="Dermatofibroma"):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        derimages.append(df["image"][i])
```

```

for i in range(len(df)):
    if(df["type"][i]=="Dermatofibroma"):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        derimages.append(df["image"][i])
        j=0
        for i in datagen1.flow(array,save_to_dir="new",save_prefix="der",save_format="jpeg"):
            j+=1
            if(j>10):
                break

```

KeyboardInterrupt

```

vasimages=[]
for i in range(len(df)):
    if(df["type"][i]=="Vascular lesions"):
        vasimages.append(df["image"][i])

for i in range(len(df)):
    if(df["type"][i]=="Vascular lesions"):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        j=0
        for i in datagen1.flow(array,save_to_dir="new1",save_prefix="vas",save_format="jpeg"):
            j+=1
            if(j>20):
                break

```

```

actimages=[]
for i in range(len(df)):
    if(df["type"][i]=="Actinic keratoses"):
        actimages.append(df["image"][i])

for i in range(len(df)):
    if(df["type"][i]=="Actinic keratoses"):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        j=0
        for i in datagen1.flow(array,save_to_dir="new2",save_prefix="act",save_format="jpeg"):
            j+=1
            if(j>10):
                break

```

```

basimages=[]
for i in range(len(df)):
    if(df["type"][i]=="Basal cell carcinoma"):
        basimages.append(df["image"][i])

```

```

for i in range(len(df)):
    if(df["type"][i]=="Basal cell carcinoma"):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        j=0
        for i in datagen1.flow(array,save_to_dir="new3",save_prefix="bas",save_format="jpeg"):
            j+=1
            if(j>5):
                break

benimages=[]
for i in range(len(df)):
    if(df["type"][i]=="Benign keratosis-like lesions "):
        benimages.append(df["image"][i])

for i in range(len(df)):
    if(df["type"][i]=="Benign keratosis-like lesions "):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        j=0
        for i in datagen1.flow(array,save_to_dir="new4",save_prefix="ben",save_format="jpeg"):
            j+=1
            if(j>1):
                break

melimages=[]
for i in range(len(df)):
    if(df["type"][i]=="Melanoma"):
        melimages.append(df["image"][i])

for i in range(len(df)):
    if(df["type"][i]=="Melanoma"):
        array=df["image"][i].reshape((1,)+df["image"][i].shape)
        j=0
        for i in datagen1.flow(array,save_to_dir="new5",save_prefix="mel",save_format="jpeg"):
            j+=1
            if(j>3):
                break

d="new1"
vasm=[]
vas=[]
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    vas.append(full_path)
    vasm.append(path)

print(x)

```

	type
Melanocytic nevi	6705
Melanoma	1113
Benign keratosis-like lesions	1099
Basal cell carcinoma	514
Actinic keratoses	327
Vascular lesions	142


```
for i in range(len(vas)):
    vasimages.append(np.asarray(Image.open(vas[i])))
```

```
d="new2"
actm=[]
act=[]
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    act.append(full_path)
    actm.append(path)
```

```
for i in range(len(act)):
    actimages.append(np.asarray(Image.open(act[i])))
```

```
d="new3"
bas=[]
basn=[]
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    bas.append(full_path)
    basn.append(path)
for i in range(len(bas)):
    basimages.append(np.asarray(Image.open(bas[i])))
```

```
d="new"
der=[]
derm=[]
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    der.append(full_path)
    derm.append(path)
print(len(derm))
```

```
for i in range(len(der)):
    derimages.append(np.asarray(Image.open(der[i])))
```

3033

```
d="new4"
ben=[]
benm=[]
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    ben.append(full_path)
    benm.append(path)
for i in range(2300):
    index=randint(0,len(ben)-1)
    benimages.append(np.asarray(Image.open(ben[index])))
```

```
import random
from random import randint
```

```

d="new5"
mel=[]
melm=[]
for path in os.listdir(d):
    full_path = os.path.join(d, path)
    mel.append(full_path)
    melm.append(path)
for i in range(2500):
    index=randint(0,len(mel)-1)
    melimages.append(np.asarray(Image.open(mel[index])))

```

```

melt=[]
for i in range(len(df)):
    if(df["type"][i]=="Melanocytic nevi"):
        melt.append(df["image"][i])
print(len(melt))

```

6705

```

import random
from random import randint

```

```

melt1=[]
for i in range(3600):
    index=randint(0,len(melt)-1)
    melt1.append(melt[index])

```

```

melt=melt1

```

```

v=[]
v.extend(melt)
print(len(v))

```

3600

```

v.extend(melimages)

```

```

v.extend(benimages)
v.extend(basimages)
v.extend(actimages)
v.extend(vasimages)
v.extend(derimages)

```

```

print(len(v))

```

22993

```

print(len(benimages),len(basimages),len(actimages),len(vasimages),len(derimages))
print(len(melimages))

```

```
3399 3161 3338 2734 3148
3613
```

```
lb6=[0 for i in range(len(melt))]
print(len(lb6))
```

```
3600
```

```
lb5=[1 for i in range(len(melimages))]
```

```
lb4=[2 for i in range(len(benimages))]
lb3=[3 for i in range(len(basimages))]
lb2=[4 for i in range(len(actimages))]
lb11=[5 for i in range(len(vasimages))]
lb10=[6 for i in range(len(derimages))]
```

```
out=[]
out.extend(lb6)
out.extend(lb5)
out.extend(lb4)
out.extend(lb3)
out.extend(lb2)
out.extend(lb11)
out.extend(lb10)
```

```
print(len(out))
```

```
22993
```

```
print(len(v))
```

```
22993
```

```
inp=[]
oup=[]
import random
from random import randint
for i in range(len(v)):
    inp.append(v[i])
    oup.append(out[i])
```

```
for i in range(len(v)):
    if(v[i]=='new2\\Actinic keratoses_0_4111.jpeg'):
        print(i)
```

```
C:\Users\shiri\anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: elementwise
```

```
inp=np.array(inp)
```

```

print(inp.shape)

(22993, 75, 100, 3)

inp=inp.reshape(22993,22500)

75*100*3
import matplotlib.pyplot as plt

10015

import numpy
import pygad
import pygad.nn
import pygad.gann

```

Start coding or [generate](#) with AI.

```

def fitness_func(solution, sol_idx):
    global GANN_instance, data_inputs, data_outputs

    predictions = pygad.nn.predict(last_layer=GANN_instance.population_networks[sol_idx],
                                   data_inputs=data_inputs)
    correct_predictions = numpy.where(predictions == data_outputs)[0].size
    solution_fitness = (correct_predictions/data_outputs.size)*100

    return solution_fitness

def callback_generation(ga_instance):
    global GANN_instance

    population_matrices = pygad.gann.population_as_matrices(population_networks=GANN_instance.popu
                                                            population_vectors=ga_instance.populat

    GANN_instance.update_population_trained_weights(population_trained_weights=population_matrices

    print("Generation = {generation}".format(generation=ga_instance.generations_completed))
    print("Accuracy   = {fitness}".format(fitness=ga_instance.best_solution()[1]))

```