Mobilenet Convolutional Architecture

In [1]:
```
#Importing all neccessary librarys
import shutil
import numpy as np
import pandas as pd
from random import random

# Image operations and plotting
from PIL import Image
from skimage.io import imread, imshow
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="whitegrid")
%matplotlib inline

# File, path and directory operations
import os
import os.path
import shutil
from glob import glob

# We are only using Keras for data augmentation
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator, img_to_arra
y, load_img

# Model building
from fastai.vision import *
from fastai.callbacks.hooks import *
import torchvision

# Data preparation
from sklearn.model_selection import train_test_split

# For reproducability
from numpy.random import seed
seed(108)

# For aesthetics
import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
```

In [2]:
```
print(os.listdir("../input"))
```

```
['skin-cancer-mnist-ham10000']
```

Loading HAM10000_metadata and splitting into train , validation and test

In [3]:
```
print(os.listdir("../input/skin-cancer-mnist-ham10000"))
```

```
['hmnist_28_28_RGB.csv', 'ham10000_images_part_1', 'HAM10000_image
s_part_2', 'hmnist_28_28_L.csv', 'HAM10000_images_part_1', 'HAM1000
0_metadata.csv', 'hmnist_8_8_RGB.csv', 'hmnist_8_8_L.csv', 'ham1000
0_images_part_2']
```

In [4]:
```
# Create a new directory
base = "base"
os.mkdir(base)
```

In [5]:
```
#[CREATE FOLDERS INSIDE THE BASE DIRECTORY]

# now we create 7 folders inside 'base':

# train
    # nv
    # mel
    # bkl
    # bcc
    # akiec
    # vasc
    # df

# valid
    # nv
    # mel
    # bkl
    # bcc
    # akiec
    # vasc
    # df

# create a path to 'base' to which we will join the names of the new f
olders
# train
train = os.path.join(base, 'train')
os.mkdir(train)

# valid
valid = os.path.join(base, 'valid')
os.mkdir(valid)


# [CREATE FOLDERS INSIDE THE TRAIN, VALIDATION AND TEST FOLDERS]
# Inside each folder we create seperate folders for each class

# create new folders inside train
nv = os.path.join(train, 'nv')
os.mkdir(nv)
mel = os.path.join(train, 'mel')
```

```
os.mkdir(mel)
bkl = os.path.join(train, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(train, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(train, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(train, 'vasc')
os.mkdir(vasc)
df = os.path.join(train, 'df')
os.mkdir(df)




# test
test = os.path.join(base, 'test')
os.mkdir(test)

nv = os.path.join(test, 'nv')
os.mkdir(nv)
mel = os.path.join(test, 'mel')
os.mkdir(mel)
bkl = os.path.join(test, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(test, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(test, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(test, 'vasc')
os.mkdir(vasc)
df = os.path.join(test, 'df')
os.mkdir(df)
```

In [6]:
```
# create new folders inside valid
nv = os.path.join(valid, 'nv')
os.mkdir(nv)
mel = os.path.join(valid, 'mel')
os.mkdir(mel)
bkl = os.path.join(valid, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(valid, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(valid, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(valid, 'vasc')
os.mkdir(vasc)
df = os.path.join(valid, 'df')
os.mkdir(df)
```

In [7]:
```
import pandas as pd
df = pd.read_csv("../input/skin-cancer-mnist-ham10000/HAM10000_metadata.csv")
```

In [8]:
```
from numpy.random import seed
seed(101)
df2=df.iloc[:,1:3]
msk = np.random.rand(len(df2)) < 0.85
train1_df2 = df2[msk]
test_df2 = df2[~msk]
msk1 = np.random.rand(len(train1_df2)) < 0.85
train_df2 = train1_df2[msk1]
validation_df2 = train1_df2[~msk1]
```

In [9]:
```
train_df2['dx'].value_counts()
```

Out[9]:
```
nv       4783
mel       846
bkl       799
bcc       378
akiec     237
vasc      101
df         80
Name: dx, dtype: int64
```

In [10]:
```
validation_df2['dx'].value_counts()
```

Out[10]:
```
nv       864
mel      125
bkl      119
bcc       70
akiec     42
df        18
vasc      17
Name: dx, dtype: int64
```

In [11]:
```
test_df2['dx'].value_counts()
```

Out[11]:
```
nv       1058
bkl       181
mel       142
bcc        66
akiec      48
vasc       24
df         17
Name: dx, dtype: int64
```

In [12]:
```
# Set the image_id as the index in df_data
df.set_index('image_id', inplace=True)
```

In [13]:
```
# Get a list of images in each of the two folders
folder_1 = os.listdir('../input/skin-cancer-mnist-ham10000/HAM10000_images_part_1')
folder_2 = os.listdir('../input/skin-cancer-mnist-ham10000/HAM10000_images_part_2')
```

In [14]:
```
# Get a list of train , val and test images

train_df2_list = list(train_df2['image_id'])
validation_df2_list = list(validation_df2['image_id'])
test_df2_list = list(test_df2['image_id'])
```

In [15]:
```
# Transfer the train images

for image in train_df2_list:

    fname = image + '.jpg'
    label = df.loc[image,'dx']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_1', fname)
        # destination path to image
        dst = os.path.join(train, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_2', fname)
        # destination path to image
        dst = os.path.join(train, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

In [16]:
```
# Transfer the val images

for image in validation_df2_list:

    fname = image + '.jpg'
    label = df.loc[image,'dx']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_1', fname)
        # destination path to image
        dst = os.path.join(valid, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_2', fname)
        # destination path to image
        dst = os.path.join(valid, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

In [17]:
```
for image in test_df2_list:

    fname = image + '.jpg'
    label = df.loc[image,'dx']

    if fname in folder_1:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_1', fname)
        # destination path to image
        dst = os.path.join(test, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
    if fname in folder_2:
        # source path to image
        src = os.path.join('../input/skin-cancer-mnist-ham10000/HAM100
00_images_part_2', fname)
        # destination path to image
        dst = os.path.join(test, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

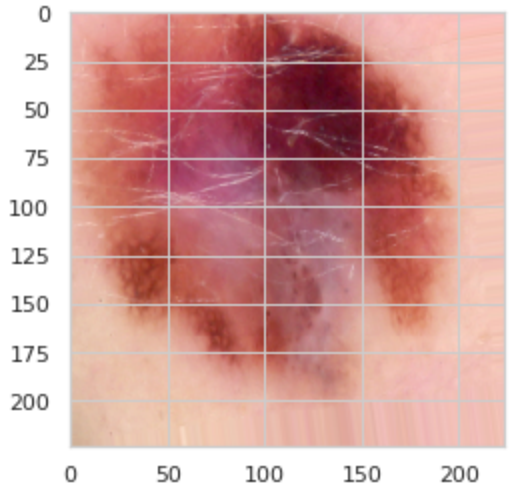
In [18]:
```
# check how many train images we have in each folder
print("............................")
print("Train folder")
print("............................")
print(len(os.listdir('base/train/nv')))
print(len(os.listdir('base/train/mel')))

print(len(os.listdir('base/train/bkl')))
print(len(os.listdir('base/train/bcc')))
print(len(os.listdir('base/train/akiec')))
print(len(os.listdir('base/train/vasc')))
print(len(os.listdir('base/train/df')))
print("............................")
# check how many train images we have in each folder
print("validation folder")
print("............................")
print(len(os.listdir('base/valid/nv')))
print(len(os.listdir('base/valid/mel')))
print(len(os.listdir('base/valid/bkl')))
print(len(os.listdir('base/valid/bcc')))
print(len(os.listdir('base/valid/akiec')))
print(len(os.listdir('base/valid/vasc')))
print(len(os.listdir('base/valid/df')))
print("............................")
# check how many train images we have in each folder
print("Test folder")
print("............................")
print(len(os.listdir('base/test/nv')))
print(len(os.listdir('base/test/mel')))
print(len(os.listdir('base/test/bkl')))
print(len(os.listdir('base/test/bcc')))
print(len(os.listdir('base/test/akiec')))
print(len(os.listdir('base/test/vasc')))
print(len(os.listdir('base/test/df')))
```

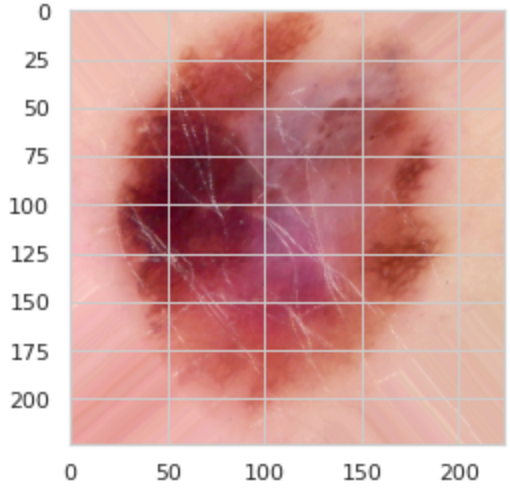
```
.............................
Train folder
.............................
4783
846
799
378
237
101
80
.............................
validation folder
.............................
864
125
119
70
42
17
18
.............................
Test folder
.............................
1058
142
181
66
48

24
17
```

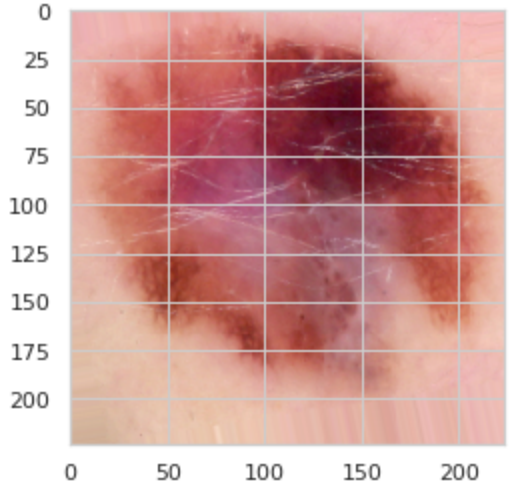
Balancing the unbalanced classes using Data Augmentation technique

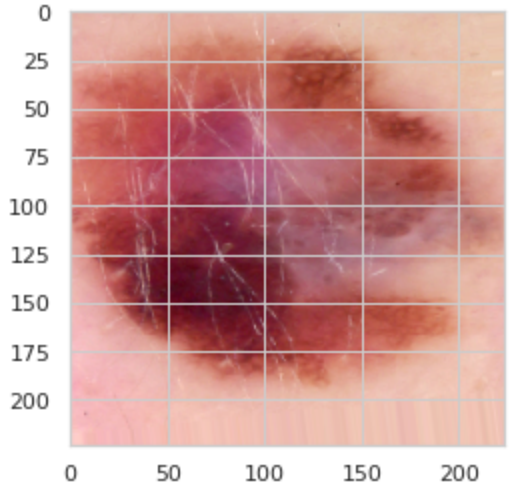
In [19]:

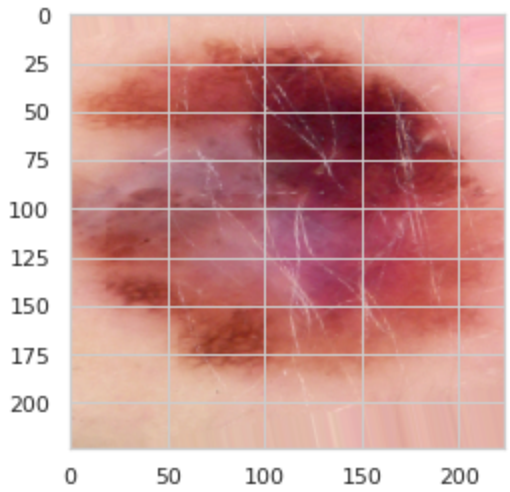
```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from tensorflow.keras.preprocessing.image import array_to_img, img_t
o_array, load_img
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=180,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest')
img_path = load_img('../input/skin-cancer-mnist-ham10000/HAM10000_imag
es_part_2/ISIC_0029316.jpg',target_size=(224, 224))
 # this is a PIL image
x = img_to_array(img_path)  # Numpy array with shape (224, 224, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 224, 224,
3)

# The .flow() command below generates batches of randomly transformed
images
# It will loop indefinitely, so we need to `break` the loop at some po
int!
i = 0
for batch in datagen.flow(x, batch_size=1):
  plt.figure(i)
  imgplot = plt.imshow(array_to_img(batch[0]))
  i += 1
  if i % 5 == 0:
    break
```




0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200
0
50
100
150
200



In [20]:
```
# note that we are not augmenting class 'nv'
class_list = ['mel','bkl','bcc','akiec','vasc','df']

for item in class_list:

    # We are creating temporary directories here because we delete the
se directories later
    # create a base
    aug = 'aug'
    os.mkdir(aug)
    # create a dir within the base to store images of the same class
    img_dir = os.path.join(aug, 'img_dir')
    os.mkdir(img_dir)

    # Choose a class
    img_class = item

    # list all images in that directory
    img_list = os.listdir('base/train/' + img_class)

    # Copy images from the class train dir to the img_dir e.g. class
'mel'
    for fname in img_list:
            # source path to image
            src = os.path.join('base/train/' + img_class, fname)
            # destination path to image
            dst = os.path.join(img_dir, fname)
            # copy the image from the source to the destination
            shutil.copyfile(src, dst)


    # point to a dir containing the images and not to the images thems
elves
    path = aug
    save_path = 'base/train/' + img_class

    # Create a data generator
    datagen = ImageDataGenerator(
        rotation_range=180,
```

```
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        vertical_flip=True,
        #brightness_range=(0.9,1.1),
        fill_mode='nearest')

    batch_size = 50

    aug_datagen = datagen.flow_from_directory(path,
                                              save_to_dir=save_path,
                                              save_format='jpg',
                                                      target_size=(224,2
24),
                                                      batch_size=batch_s
ize)



    # Generate the augmented images and add them to the training folde
rs

    ###########

    num_aug_images_wanted = 5000 # total number of images we want to h
ave in each class

    ###########

    num_files = len(os.listdir(img_dir))
    num_batches = int(np.ceil((num_aug_images_wanted-num_files)/batc
h_size))

    # run the generator and create about 6000 augmented images
    for i in range(0,num_batches):

        imgs, labels = next(aug_datagen)

    # delete temporary directory with the raw image files
    shutil.rmtree('aug')
```

```
Found 846 images belonging to 1 classes.
Found 799 images belonging to 1 classes.
Found 378 images belonging to 1 classes.
Found 237 images belonging to 1 classes.
Found 101 images belonging to 1 classes.
Found 80 images belonging to 1 classes.
```

In [21]:
```
# check how many train images we have in each folder
print("...............................")
print("Train folder")
print("...............................")
print(len(os.listdir('base/train/nv')))
print(len(os.listdir('base/train/mel')))
print(len(os.listdir('base/train/bkl')))
print(len(os.listdir('base/train/bcc')))
print(len(os.listdir('base/train/akiec')))
print(len(os.listdir('base/train/vasc')))
print(len(os.listdir('base/train/df')))
print("...............................")
# check how many train images we have in each folder
print("validation folder")
print("...............................")
print(len(os.listdir('base/valid/nv')))
print(len(os.listdir('base/valid/mel')))
print(len(os.listdir('base/valid/bkl')))
print(len(os.listdir('base/valid/bcc')))
print(len(os.listdir('base/valid/akiec')))
print(len(os.listdir('base/valid/vasc')))
print(len(os.listdir('base/valid/df')))
print("...............................")
# check how many train images we have in each folder
print("Test folder")
print("...............................")
print(len(os.listdir('base/test/nv')))
print(len(os.listdir('base/test/mel')))
print(len(os.listdir('base/test/bkl')))
print(len(os.listdir('base/test/bcc')))
print(len(os.listdir('base/test/akiec')))
print(len(os.listdir('base/test/vasc')))
print(len(os.listdir('base/test/df')))
```

```
.............................
Train folder
.............................
4783
5030
5044
4786
4790
3433
4050
.............................
validation folder
.............................
864
125
119
70
42
17
18
.............................
Test folder
.............................
1058
142
181
66
48
24

17
```

Using MobileNet Architecture

In [22]:
```
# Define transformations for data augmentation
tfms = get_transforms(do_flip=True,
                      max_rotate=10,
                      max_zoom=1.1,
                      max_warp=0.2)

# Build dataset by applying transforms to the data from our directory
data = (ImageList.from_folder(base)
        .split_by_folder()
        .label_from_folder()
        .transform(tfms, size=224)
        .databunch()
        .normalize(imagenet_stats))
```

In [23]:
```
wd=1e-2

mobilenet_split = lambda m: (m[0][0][10], m[1])
arch  = torchvision.models.mobilenet_v2
learn = cnn_learner(data, arch, cut=-1, split_on=mobilenet_split, wd=w
d, metrics=[accuracy])
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353
104.pth" to /root/.cache/torch/checkpoints/mobilenet_v2-b0353104.pth
```

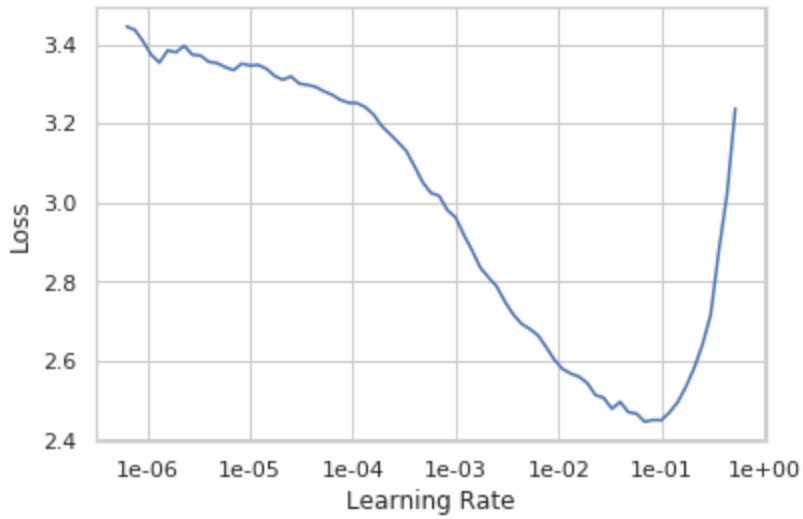
In [24]:
```
learn.lr_find();
learn.recorder.plot();
```

0.00% [0/1 00:00<00:00]

epoch	train_loss	valid_loss	accuracy	time

17.87% [89/498 01:07<05:12 8.3212]

```
LR Finder is complete, type {learner_name}.recorder.plot() to see th
e graph.
```



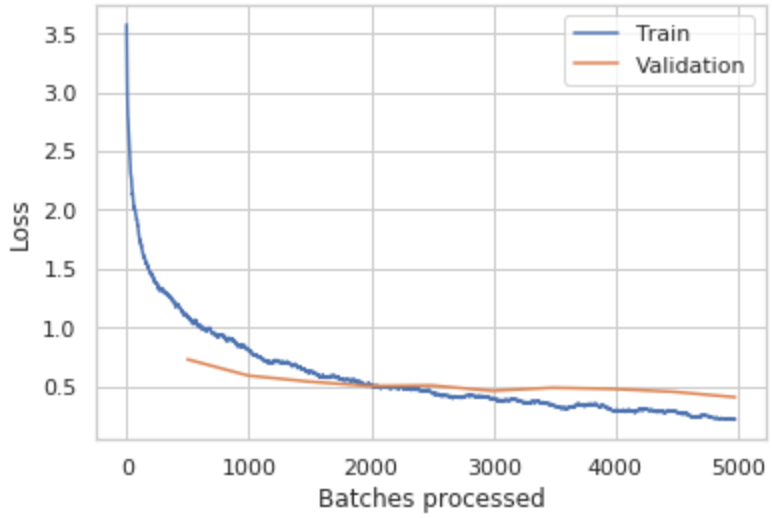
In [25]:
```
# Set our learning rate to the value where learning is fastest and los
s
# is still decreasing.
lr=3e-3

# This function uses our input lr as an anchor and sweeps through a ra
nge
# in order to search out the best local minima.
learn.fit_one_cycle(10, slice(lr), pct_start=0.9)
```

epoch	train_loss	valid_loss	accuracy	time
0	1.088360	0.730347	0.778486	06:41
1	0.807788	0.592182	0.807171	06:48
2	0.627882	0.541020	0.811155	06:53
3	0.516480	0.503540	0.812749	06:59
4	0.453764	0.509392	0.816733	06:58
5	0.394898	0.462734	0.831076	06:59
6	0.341926	0.489190	0.829482	07:09
7	0.293263	0.478801	0.837450	06:50
8	0.295792	0.454103	0.837450	06:40
9	0.221239	0.408573	0.861355	06:44

In [26]:
```
# Save the current state of our model
learn.save('mobile_v1_stage-1')
```

Plot of training and validation loss

In [27]:
```
learn.recorder.plot_losses()
```



In [28]:
```
# Exctract predictions and losses to evaluate model

preds,y,losses = learn.get_preds(with_loss=True)
interp = ClassificationInterpretation(learn, preds, y, losses)
```

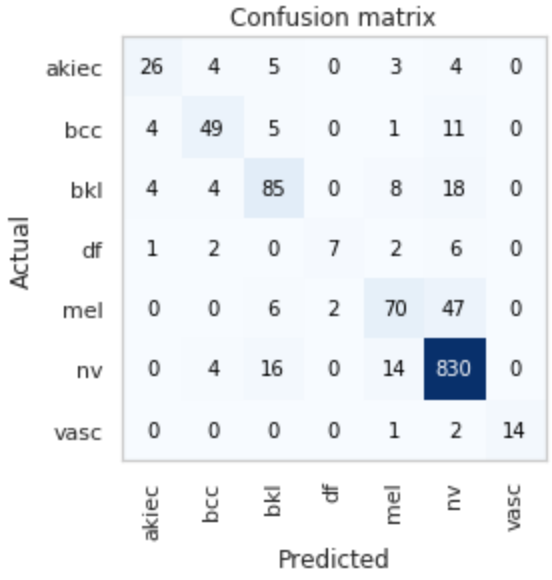
In [29]:
```
def top_k_spread(preds, y, spread):
  for i in range(spread):
    print(f"Top {i+1} accuracy: {top_k_accuracy(preds, y, i+1)}")
```

In [30]:
```
# Top-1 accuracy of 86% is quite near the best models from the open co
mpetition
top_k_spread(preds, y, 5)
```

```
Top 1 accuracy: 0.8613545894622803
Top 2 accuracy: 0.9505975842475891
Top 3 accuracy: 0.9832669496536255
Top 4 accuracy: 0.9920318722724915
Top 5 accuracy: 0.9968127608299255
```

Confusion Matrix Report

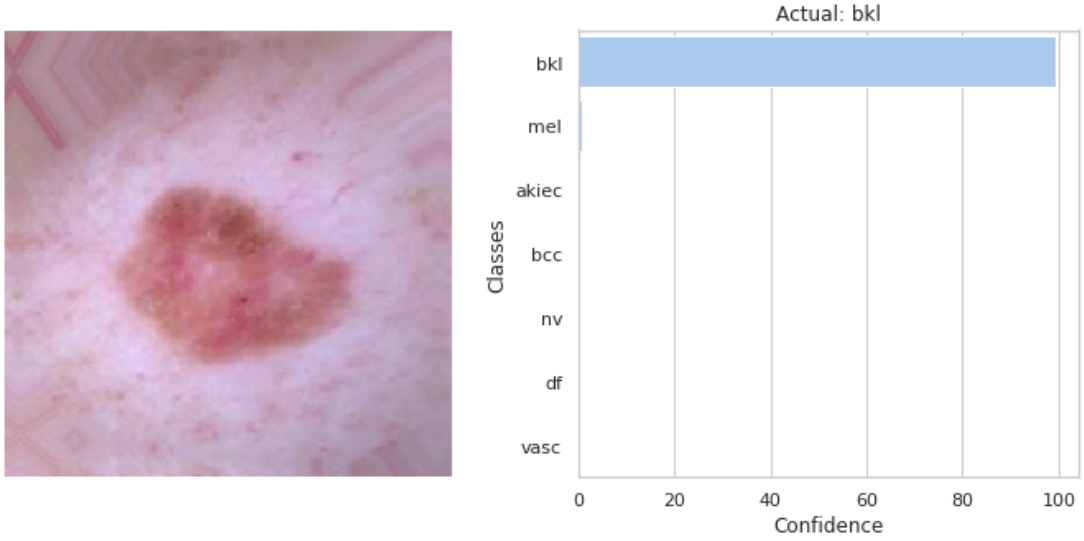
In [31]:
```
interp.plot_confusion_matrix()
```



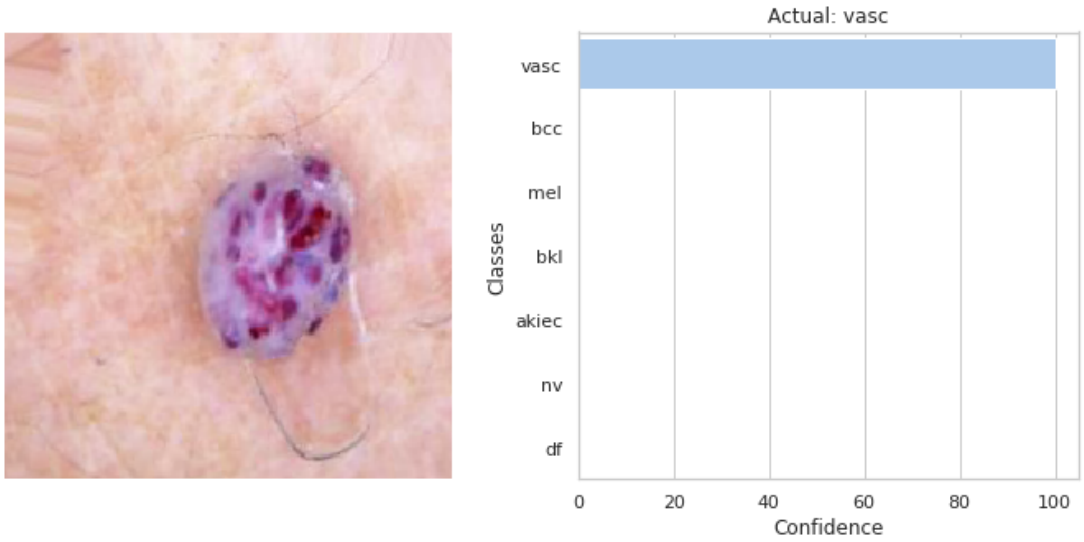
In [32]:
```
# This function creates a display of the models prediction and confide
nce levels
def plot_prediction(learner, index):
  data = learner.data.train_ds[index][0]
  pred = learner.predict(data)
  classes = learner.data.classes

  prediction = pd.DataFrame(to_np(pred[2]*100), columns=['Confidenc
e'])
  prediction['Classes'] = classes
  prediction = prediction.sort_values(by='Confidence', ascending=Fals
e)

  fig = plt.figure(figsize=(12, 5))
  ax1 = fig.add_subplot(121)
  show_image(data, figsize=(5, 5), ax=ax1)
  ax2 = fig.add_subplot(122)
  sns.set_color_codes("pastel")
  sns.barplot(x='Confidence', y='Classes', data=prediction,
              label="Total", color="b")
  ax2.set_title(f'Actual: {learn.data.train_ds[index][1]}')
```

Predictions of some random images

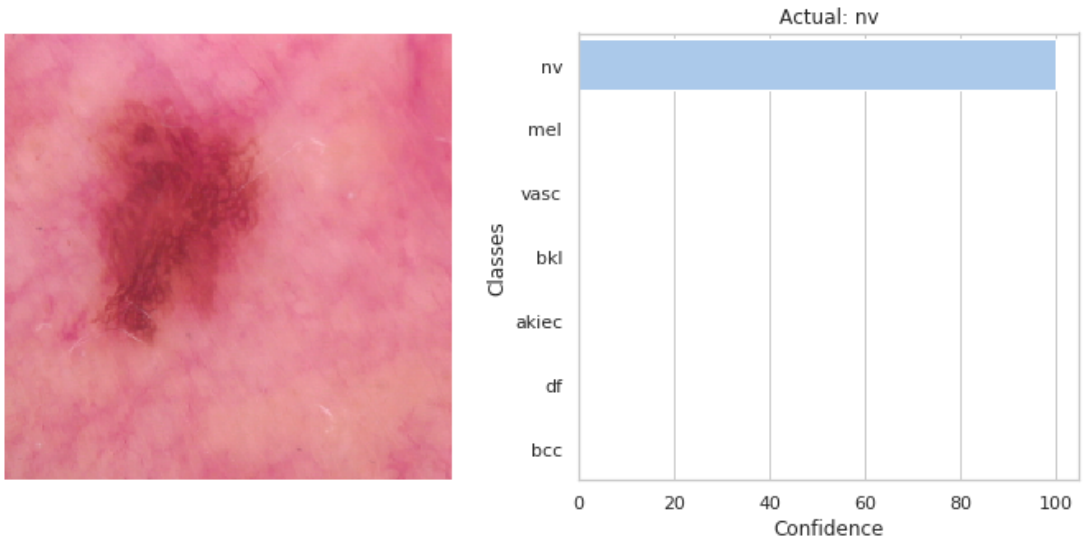
In [33]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



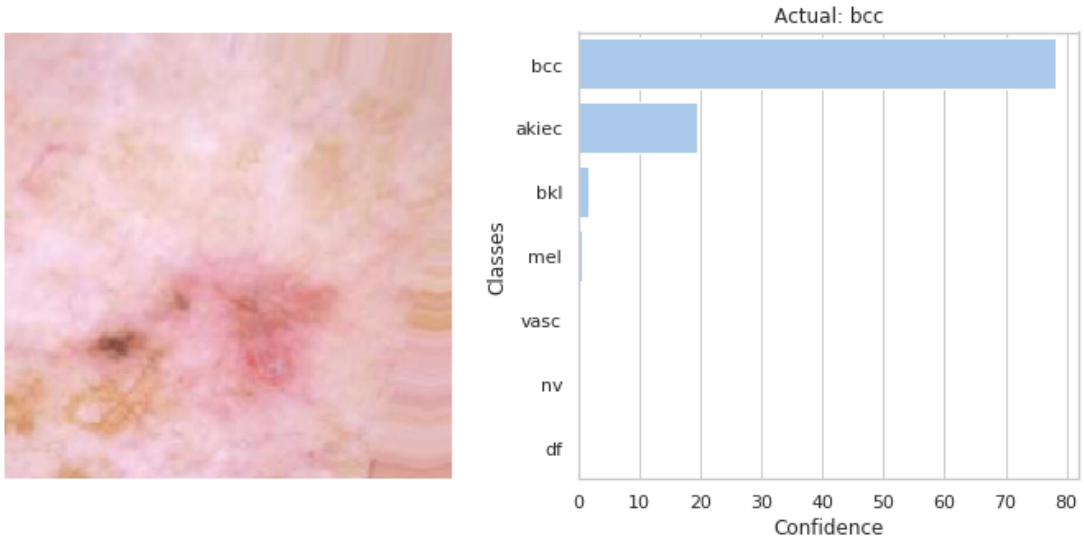
In [34]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



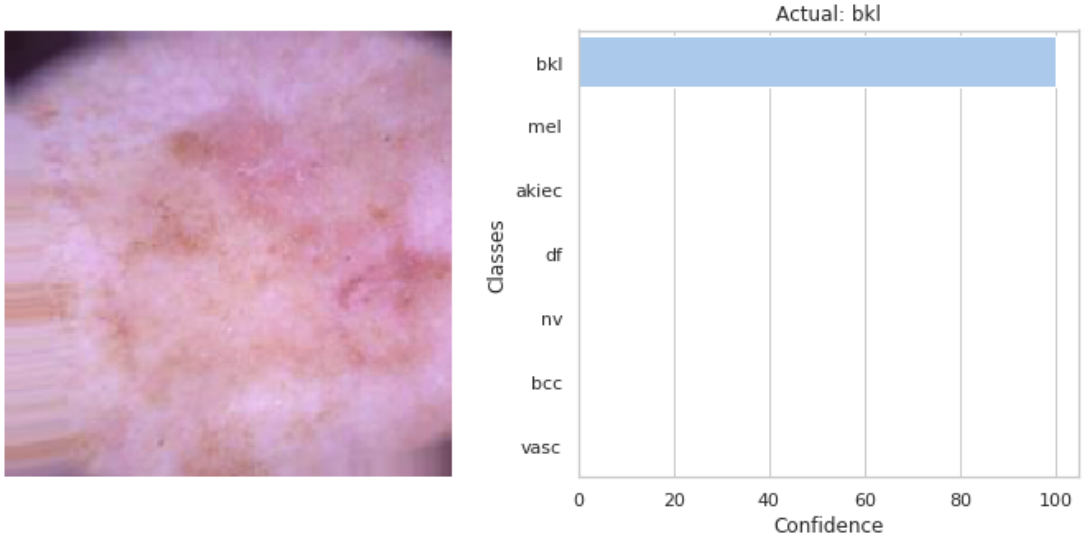
In [35]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



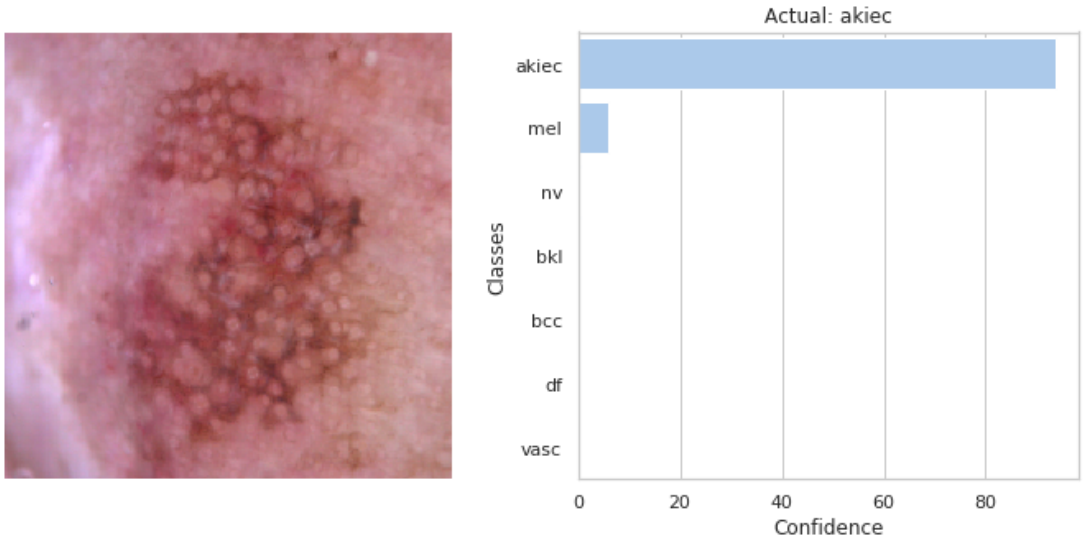
In [36]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



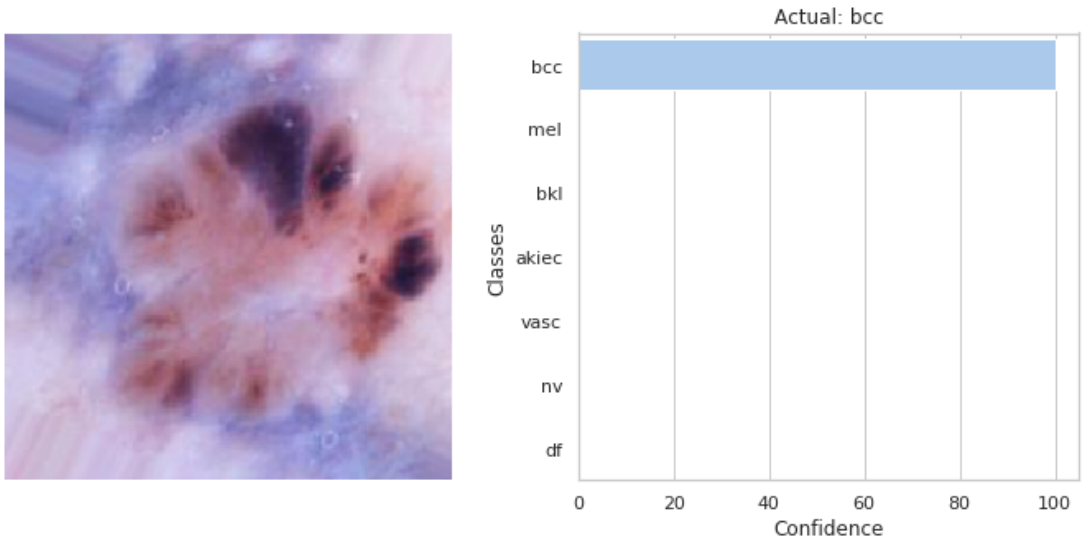
In [37]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



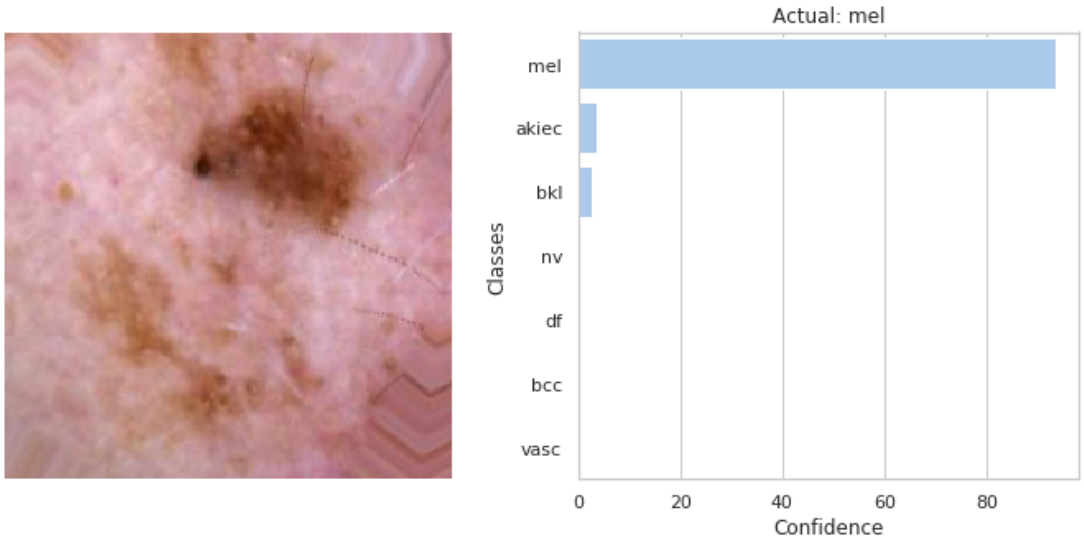
In [38]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



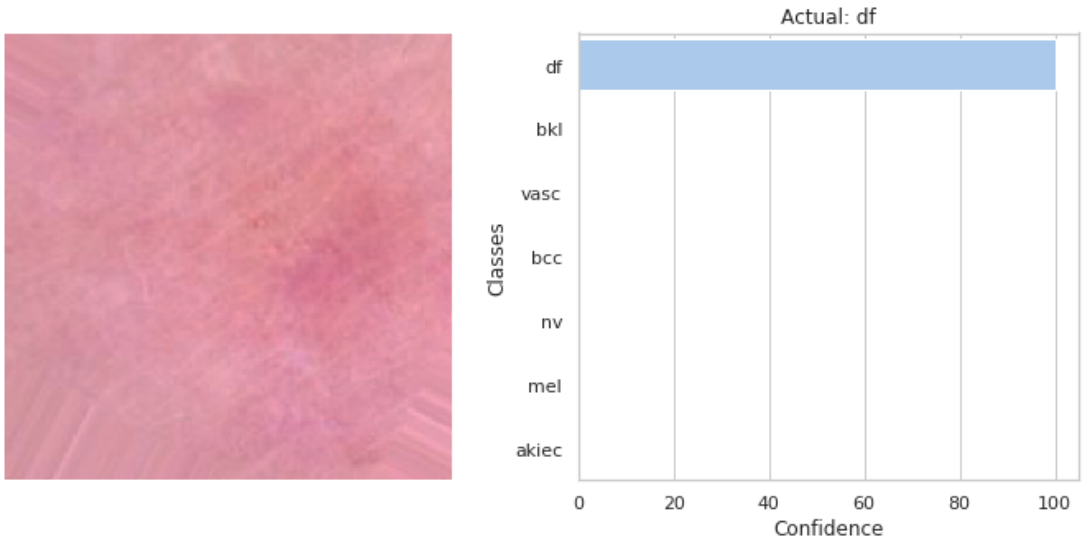
In [39]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



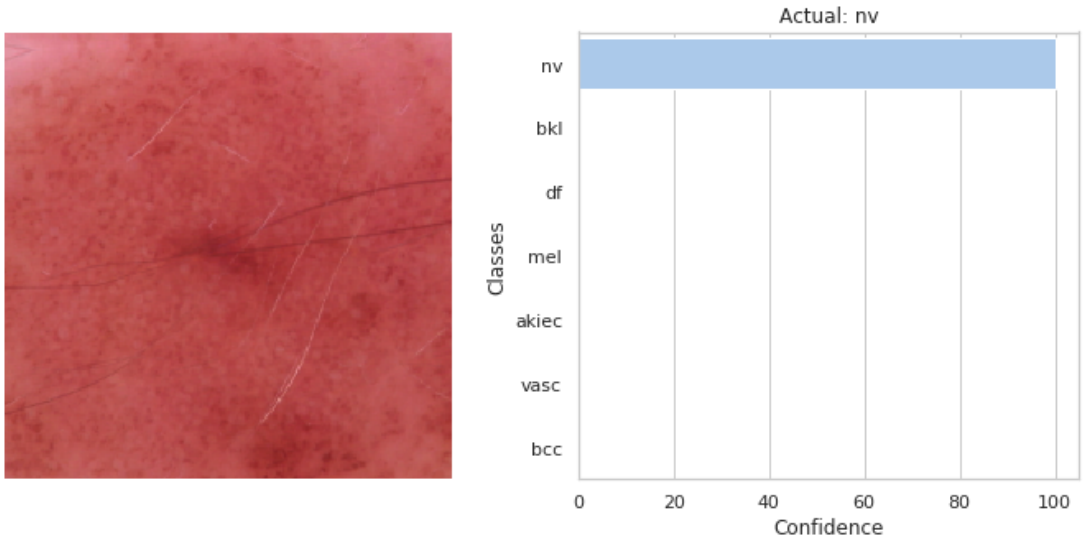
In [40]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



In [41]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



In [42]:
```
plot_prediction(learn, np.random.choice(len(learn.data.train_ds)))
```



In [ ]:
```
```