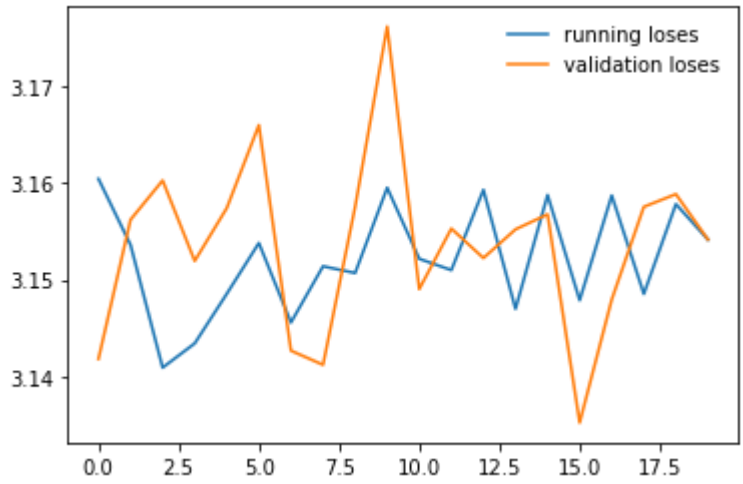
```
<matplotlib.legend.Legend at 0x7f1ee9c27400>
```



Visualise the predictions

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(testloader):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            # _, preds = torch.max(outputs, 1)
            ps = torch.exp(outputs)
            top_p, top_class = ps.topk(1, dim=1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('predicted: {}'.format(class_names[top_class[j]]))
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

```
visualize_model(model, num_images=10)
```

predicted: Body (ringworm)

predicted: Acne Excoriated

predicted: Incognito (ringworm)

predicted: Scalp (ringworm)

predicted: Vasculitis

predicted: Face (ringworm)

predicted: Foot (ringworm)

predicted: Palm (ringworm)

predicted: Palm (ringworm)

predicted: Palm (ringworm)

```
from sklearn.metrics import confusion_matrix
class_correct = list(0. for i in range(26))
class_total = list(0. for i in range(26))
results = None
with torch.no_grad():
    for inputs, labels in testloader:
      inputs, labels = inputs.to(device), labels.to(device)
      outputs = model(inputs)

      _, predicted = torch.max(outputs, 1)
      c = (predicted == labels).squeeze()
      results = confusion_matrix(labels.cpu(), predicted.cpu())
      for i in range(4):
          label = labels[i]
```

```
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(26):
    print('Accuracy of %5s : %2d %%' % (
        class_names[i], 100 * class_correct[i] / class_total[i]))
print(results)
```

```
Accuracy of  AIDS :  0 %
Accuracy of Acne Closed Comedo :  0 %
Accuracy of Acne Cystic :  0 %
Accuracy of Acne Excoriated : 50 %
Accuracy of Acne Infantile :  0 %
Accuracy of Acne Open Comedo :  0 %
Accuracy of Acne Pustular :  0 %
Accuracy of Acne Scar :  0 %
Accuracy of Body (ringworm) :  0 %
Accuracy of Eczema Herpeticum :  0 %
Accuracy of Face (ringworm) : 33 %
Accuracy of Foot (ringworm) :  0 %
Accuracy of Foot Plantar (ringworm) :  0 %
Accuracy of Foot Webs (ringworm) :  0 %
Accuracy of Genital Warts :  0 %
Accuracy of Groin (ringworm) :  0 %
Accuracy of Hand Dorsum (ringworm) :  0 %
Accuracy of Incognito (ringworm) : 25 %
Accuracy of Palm (ringworm) : 100 %
Accuracy of Primary Lesion (ringworm) :  0 %
Accuracy of Scabies :  0 %
Accuracy of Scalp (ringworm) : 80 %
Accuracy of Vasculitis :  0 %
Accuracy of Versicolor (ringworm) : 40 %
Accuracy of Warts Common :  0 %
Accuracy of Warts Plantar :  0 %
[[0 0 0 0 0 0 0 0 0 0 2 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 2 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0 0 1 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 1 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]]
```

```
def process_image(image):
```

```
def process_image(image):
    ''' Scales, crops, and normalizes a PIL image for a PyTorch model,
        returns an Numpy array
    '''

    # TODO: Process a PIL image for use in a PyTorch model

    # here, we resize to make the shoter side be 256
    width, height = image.size # get dimensions
    if height > width:
        h = (height/width)*256
        image.thumbnail((h,256))
    elif height < width:
        w = (width/height)*256
        image.thumbnail((256,w))

    # center crop
    left = (width-224)/2
    top = (height-224)/2
    right = (width+224)/2
    bottom = (height+224)/2

    croped_image = image.crop((left,top,right, bottom))



    # convert image to floats between 0,1
    np_image = np.array(croped_image)
    np_image.astype(float)
    np_image = np_image/255

    # normalise image
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    np_image = (np_image-mean)/std

    # transpose image
    npt_image = np_image.transpose()
    return torch.Tensor(npt_image)
```

```
def imshow(image, ax=None, title=None):
    """Imshow for Tensor."""
    if ax is None:
        fig, ax = plt.subplots()

    # PyTorch tensors assume the color channel is the first dimension
    # but matplotlib assumes is the third dimension
    image = np.array(image).transpose((1, 2, 0))

    # Undo preprocessing
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    image = std * image + mean
```

```
    image = std * image + mean

    # Image needs to be clipped between 0 and 1 or it looks like noise when displayed
    image = np.clip(image, 0, 1)

    ax.imshow(image.squeeze())

    return ax
```

```
def predict(image_path, model, topk=5):
    ''' Predict the class (or classes) of an image using a trained deep learning model.
    '''

    # TODO: Implement the code to predict the class from an image file
    image = Image.open(image_path)
    img = process_image(image)

    # peform the model classification
    with torch.no_grad():
        model.to('cpu')
        model.eval()
        logs = model(img.unsqueeze_(0))
        ps = torch.exp(logs)

        #get the top probalities
        top_p, top_class = ps.topk(topk, dim=1)

    classes_idx = []
    top_class = top_class.squeeze()
    for i, x in enumerate(class_names):
        if i in top_class:
            classes_idx.append(i)

    return top_p, classes_idx
```

```
!wget https://www.dropbox.com/s/slcdbh712v5vzny/scabies.jpg?dl=0 -O image2.jpg
```

```
--2020-03-01 15:34:22--  https://www.dropbox.com/s/slcdbh712v5vzny/scabies.jpg?dl=0
Resolving www.dropbox.com (www.dropbox.com)... 162.125.1.1, 2620:100:6016:1::a27d:101
Connecting to www.dropbox.com (www.dropbox.com)|162.125.1.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/slcdbh712v5vzny/scabies.jpg [following]
--2020-03-01 15:34:22--  https://www.dropbox.com/s/raw/slcdbh712v5vzny/scabies.jpg
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc419f66040bd24764f6aed1c80c.dl.dropboxusercontent.com/cd/0/inline/AzH
--2020-03-01 15:34:23--  https://uc419f66040bd24764f6aed1c80c.dl.dropboxusercontent.com/
Resolving uc419f66040bd24764f6aed1c80c.dl.dropboxusercontent.com (uc419f66040bd24764f6ae
Connecting to uc419f66040bd24764f6aed1c80c.dl.dropboxusercontent.com (uc419f66040bd24764
HTTP request sent, awaiting response... 200 OK
Length: 40260 (39K) [image/jpeg]
Saving to: ‘image2.jpg’

image2.jpg          100%[===================>]  39.32K  --.-KB/s    in 0.04s

2020-03-01 15:34:23 (1023 KB/s) - ‘image2.jpg’ saved [40260/40260]
```

```
image_path = './image2.jpg'
```

```
processed_image = process_image(Image.open(image_path))
```

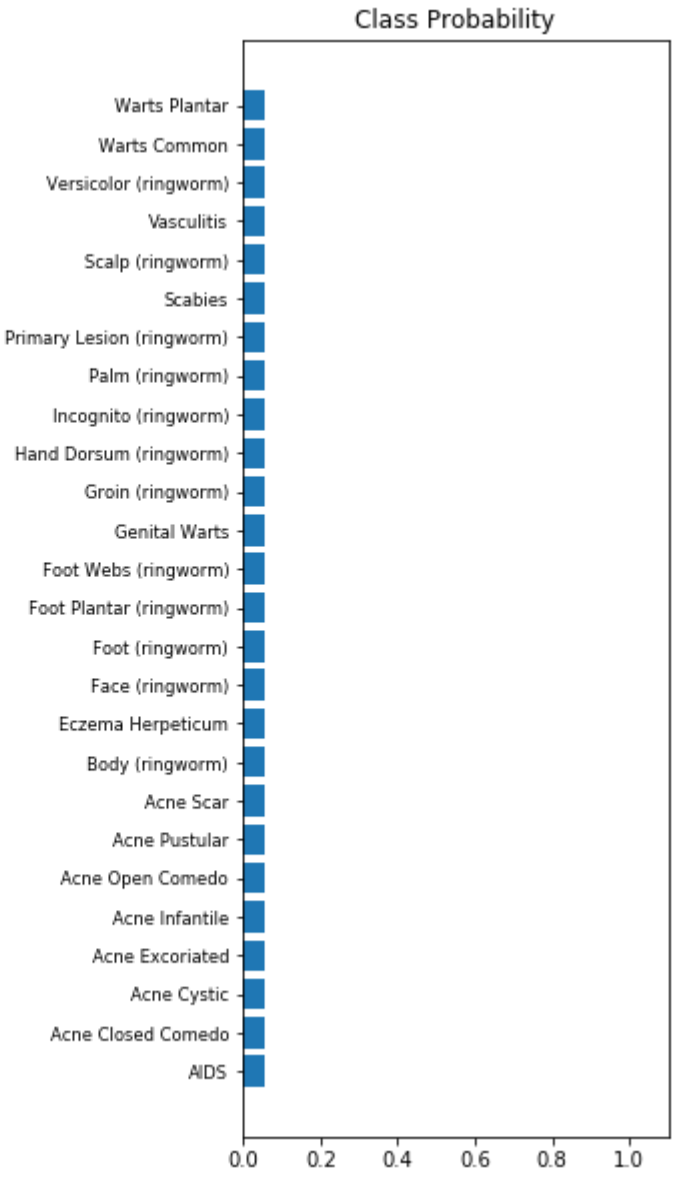
```
probs, classes = predict(image_path, model, topk=1)
```

```
def view_classify(img, ps, labels):
    ''' Function for viewing an image and it's predicted classes.
    '''
    ps = ps.squeeze()

    fig, (ax1, ax2) = plt.subplots(figsize=(8,11), ncols=2)
    # im = np.array(img.resize_((28, 28)))
    # ax1.imshow(img.squeeze())
    ax1.axis('off')
    imshow(img, ax=ax1)
    ax2.barh(np.arange(len(labels)), ps)
    ax2.set_aspect(0.1)
    ax2.set_yticks(np.arange(len(labels)))
    ax2.set_yticklabels(labels, size='small');
    ax2.set_title('Class Probability')
    ax2.set_xlim(0, 1.1)

    plt.tight_layout()
```

```
view_classify(processed_image, probs, class_names)
```



Saving and loading trained models

After trainig the model, it is important to save the current state parameters and all other parameters t
and the hyperparameters.

```
checkpoint_dir = 'checkpoints/'
!mkdir checkpoints
!ls
```

```
# crete custom dictionary to save additional params
checkpoint = {'epoch': e,
              'learning_rate': learning_rate,
              'arch': 'cpu',
              'optimizer_state_dict': optimizer.state_dict(),
             'model_state_dict': model.state_dict(),
```