University of Minho
School of Engineering

ISLab

Aprendizagem Profunda

Deep Neural Network – CNN Feature Maps visualization

AP @ MEI/1º ano – 2º Semestre

Victor Alves

Part IV

Hands On

CNN for multiclass image classification

Filter and featuremaps visualization

0. Prepare the setup
    - Install pytorch (if needed)
    - Imports
    - Constants
    - Device management (optional)
1. Prepare the data

1.1 Visualize the data

2. Define the model

3. Train the model

4. Evaluate the model

5. Use the model

```
def img_show(img, legenda):
    img=img.cpu()
    plt.axis('off')
    plt.title(legenda)
    plt.grid(b=None)
    plt.imshow(img[0,0], cmap=plt.get_cmap('gray'))
    plt.show()

def make_prediction(model, img):
    img = img.reshape(1, 1, 28, 28)
    print(img.shape)
    print(img.dtype)
    img = img.to(device)
    prediction = model(img).cpu().detach().numpy()[0].argmax()
    legenda=f"predict:{prediction}"
    img_show(img,legenda)
    return prediction
```

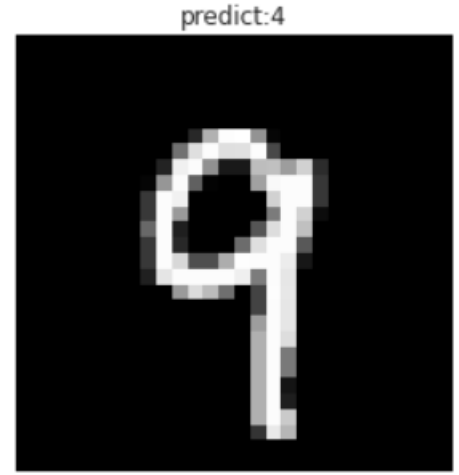
```
model= torch.load('CNNModel_1.pth')
model.eval()
imagens, label = next(iter(test_dl))
make_prediction(model,imagens[3])
```

5. Use the model

```
def img_show(img, legenda):
    img=img.cpu()
    plt.axis('off')
    plt.title(legenda)
    plt.grid(b=None)
    plt.imshow(img[0,0], cmap=plt.get_cmap('gray'))
    plt.show()

def make_prediction(model, img):
    img = img.reshape(1, 1, 28, 28)
    print(img.shape)
    print(img.dtype)
    img = img.to(device)
    prediction = model(img).cpu().detach().numpy()[0].argmax()
    legenda=f"predict:{prediction}"
    img_show(img,legenda)
    return prediction
```

```
model= torch.load('CNNModel_1.pth')
model.eval()
imagens, label = next(iter(test_dl))
make_prediction(model,imagens[3])
```

```
torch.Size([1, 1, 28, 28])
torch.float32
```




```
4
```

5. Use the model

```
def show_batch_images(model,dataloader,num_image):
    imagens, label = next(iter(test_dl))
    pred = make_prediction(model,imagens[3])
    print("pred:",pred)
    return imagens, pred

model= torch.load('CNNModel_1.pth')
model.eval()
images, pred = show_batch_images(model,test_dl,1)
print(pred)
```

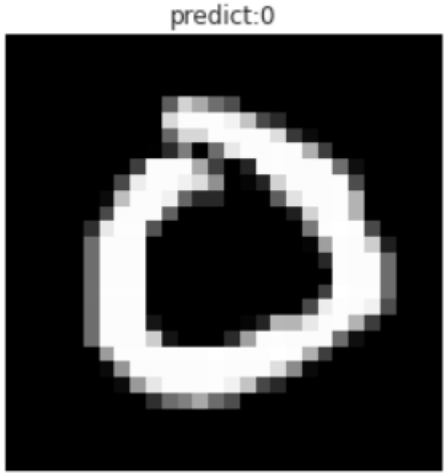
```
print(model)
```

5. Use the model

```
def show_batch_images(model,dataloader,num_image):
    imagens, label = next(iter(test_dl))
    pred = make_prediction(model,imagens[3])
    print("pred:",pred)
    return imagens, pred
```

```
model= torch.load('CNNModel_1.pth')
model.eval()
images, pred = show_batch_images(model,test_dl,1)
print(pred)
```

```
torch.Size([1, 1, 28, 28])
torch.float32
```




```
pred: 0
0
```

```
print(model)
```

```
CNNModel_1(
  (layer1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=800, out_features=100, bias=True)
  (act1): ReLU()
  (fc2): Linear(in_features=100, out_features=10, bias=True)
  (act2): Softmax(dim=1)
)
```

5. Use the model

```
def plot_filters_single_channel_big(t):
    nrows = t.shape[0]*t.shape[2]
    ncols = t.shape[1]*t.shape[3]
    npimg = np.array(t.numpy(), np.float32)
    npimg = npimg.transpose((0, 2, 1, 3))
    npimg = npimg.ravel().reshape(nrows, ncols)
    npimg = npimg.T
    fig, ax = plt.subplots(figsize=(ncols/10, nrows/200))
    imgplot = sns.heatmap(npimg, xticklabels=False, yticklabels=False, cmap='gray', ax=ax, cbar=False)
```

5. Use the model

```
def plot_filters_single_channel(t):
    nplots = t.shape[0]*t.shape[1]
    ncols = 12
    nrows = 1 + nplots//ncols
    npimg = np.array(t.numpy(), np.float32)
    count = 0
    fig = plt.figure(figsize=(ncols, nrows))
    for i in range(t.shape[0]):
        for j in range(t.shape[1]):
            count += 1
            ax1 = fig.add_subplot(nrows, ncols, count)
            npimg = np.array(t[i, j].numpy(), np.float32)
            npimg = (npimg - np.mean(npimg)) / np.std(npimg)
            npimg = np.minimum(1, np.maximum(0, (npimg + 0.5)))
            ax1.imshow(npimg)
            (...)
            (...)
            ax1.set_title(str(i) + ',' + str(j))
            ax1.axis('off')
            ax1.set_xticklabels([])
            ax1.set_yticklabels([])
    plt.tight_layout()
    plt.show()
```

5. Use the model

```
def plot_filters_multi_channel(t):
    num_kernels = t.shape[0]
    num_cols = 12
    num_rows = num_kernels
    fig = plt.figure(figsize=(num_cols,num_rows))
    for i in range(t.shape[0]):
        ax1 = fig.add_subplot(num_rows,num_cols,i+1)
        npimg = np.array(t[i].numpy(), np.float32)
        npimg = (npimg - np.mean(npimg)) / np.std(npimg)
        npimg = np.minimum(1, np.maximum(0, (npimg + 0.5)))
        npimg = npimg.transpose((1, 2, 0))
        ax1.imshow(npimg)
        ax1.axis('off')
        ax1.set_title(str(i))
        ax1.set_xticklabels([])
        ax1.set_yticklabels([])
    plt.savefig('kernels.png', dpi=100)
    plt.tight_layout()
    plt.show()
```

5. Use the model

```
def plot_weights(layer, single_channel = True, collated = False):
    if isinstance(layer, nn.Conv2d):
        weight_tensor = layer.weight.data
        if single_channel:
            if collated:
                plot_filters_single_channel_big(weight_tensor.cpu())
            else:
                plot_filters_single_channel(weight_tensor.cpu() )
        else:
            if weight_tensor.shape[1] == 3:
                plot_filters_multi_channel(weight_tensor.cpu())
            else:
                print("Can only plot weights with three channels with single channel = False")
    else:
        print("Can only visualize layers which are convolutional")
```

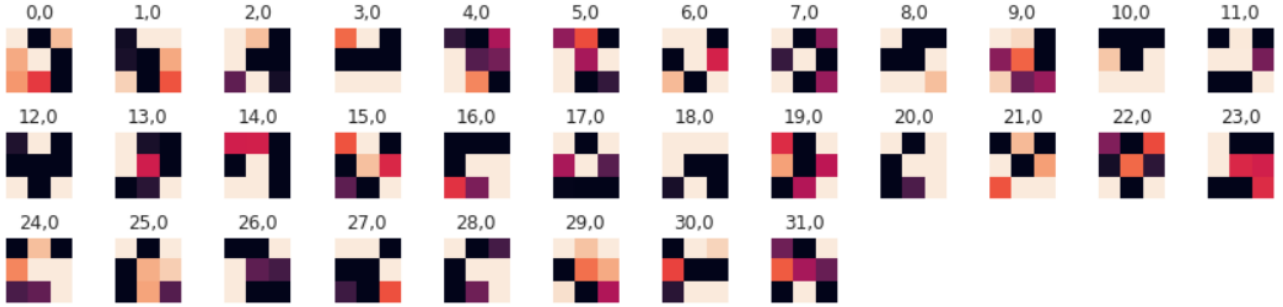
5. Use the model

```
plot_weights(model.layer1[0], single_channel = True)
```

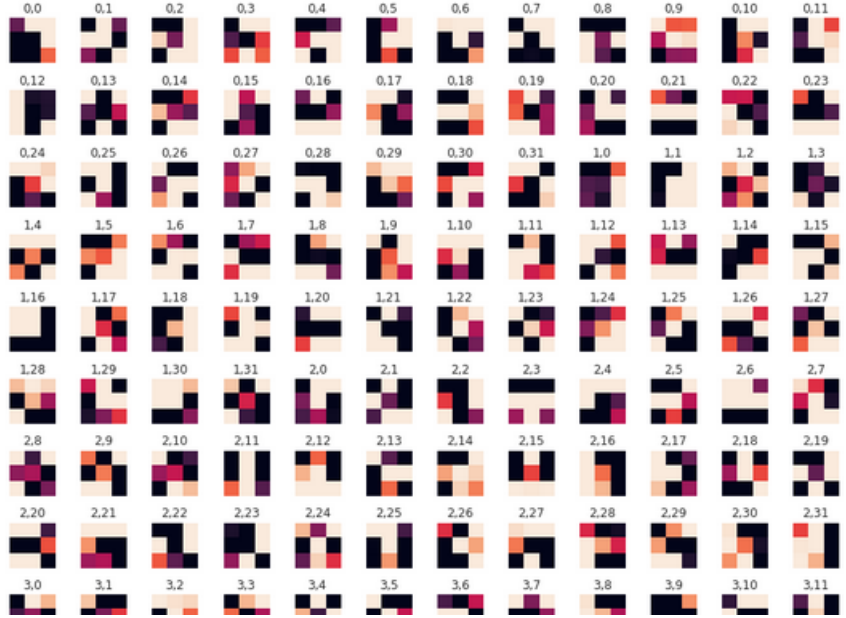
```
plot_weights(model.layer2[0], single_channel = True)
```

5. Use the model

```
plot_weights(model.layer1[0], single_channel = True)
```




```
plot_weights(model.layer2[0], single_channel = True)
```



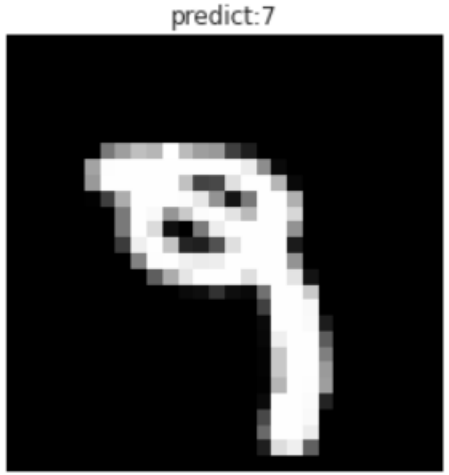
5. Use the model

```
def show_batch_images(model,dataloader,num_image):
    imagens, label = next(iter(test_dl))
    pred = make_prediction(model,imagens[3])
    print("pred:",pred)
    return imagens, pred
images, pred = show_batch_images(model,test_dl,1)
print(pred)

print(images.shape)
```

5. Use the model

```
def show_batch_images(model,dataloader,num_image):
    imagens, label = next(iter(test_dl))
    pred = make_prediction(model,imagens[3])
    print("pred:",pred)
    return imagens, pred
images, pred = show_batch_images(model,test_dl,1)
print(pred)

print(images.shape)
```

```
torch.Size([1, 1, 28, 28])
torch.float32
```




```
pred: 7
7
```

```
torch.Size([32, 1, 28, 28])
```

5. Use the model

```
def get_conv_layers(model):
    conv_layers=list()
    model_children=list(model.children())
    for child in model_children:
        if type(child)==Conv2d:
            conv_layers.append(child)
        elif type(child)==Sequential:
            for layer in child.children():
                if type(layer)==Conv2d:
                    conv_layers.append(layer)
    return conv_layers
```

5. Use the model

```
def get_conv_pool_layers(model):
    conv_layers=list()
    model_children=list(model.children())
    for child in model_children:
        if type(child)==Conv2d or type(child)==MaxPool2d:
            conv_layers.append(child)
        elif type(child)==Sequential:
            for layer in child.children():
                if type(layer)==Conv2d or (type(layer)==MaxPool2d):
                    conv_layers.append(layer)
    return conv_layers

conv_layers = get_conv_pool_layers(model)
print(conv_layers)
print(len(conv_layers))
```

5. Use the model

```
def get_conv_pool_layers(model):
    conv_layers=list()
    model_children=list(model.children())
    for child in model_children:
        if type(child)==Conv2d or type(child)==MaxPool2d:
            conv_layers.append(child)
        elif type(child)==Sequential:
            for layer in child.children():
                if type(layer)==Conv2d or (type(layer)==MaxPool2d):
                    conv_layers.append(layer)
    return conv_layers

conv_layers = get_conv_pool_layers(model)
print(conv_layers)
print(len(conv_layers))
```

```
[Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1)), MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False), Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1)), MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)]
4
```

5. Use the model

```
def percorrer_conv_layers(images, conv_layers):
    images = images.to(device)
    results = [conv_layers[0](images)]
    for i in range(1, len(conv_layers)):
        results.append(conv_layers[i](results[-1]))
    outputs = results
    return outputs

outputs = percorrer_conv_layers(images, conv_layers)
print(f"Obtiveram-se {len(outputs)} tensores com o shape:")
for i in range(len(outputs)):
    print(f"    Layer {i} - {outputs[i].shape}")
```

5. Use the model

```
def percorrer_conv_layers(images, conv_layers):
    images = images.to(device)
    results = [conv_layers[0](images)]
    for i in range(1, len(conv_layers)):
        results.append(conv_layers[i](results[-1]))
    outputs = results
    return outputs

outputs = percorrer_conv_layers(images, conv_layers)
print(f"Obtiveram-se {len(outputs)} tensores com o shape:")
for i in range(len(outputs)):
    print(f"    Layer {i} - {outputs[i].shape}")
```

```
Obtiveram-se 4 tensores com o shape:
    Layer 0 - torch.Size([32, 32, 26, 26])
    Layer 1 - torch.Size([32, 32, 13, 13])
    Layer 2 - torch.Size([32, 32, 11, 11])
    Layer 3 - torch.Size([32, 32, 5, 5])
```

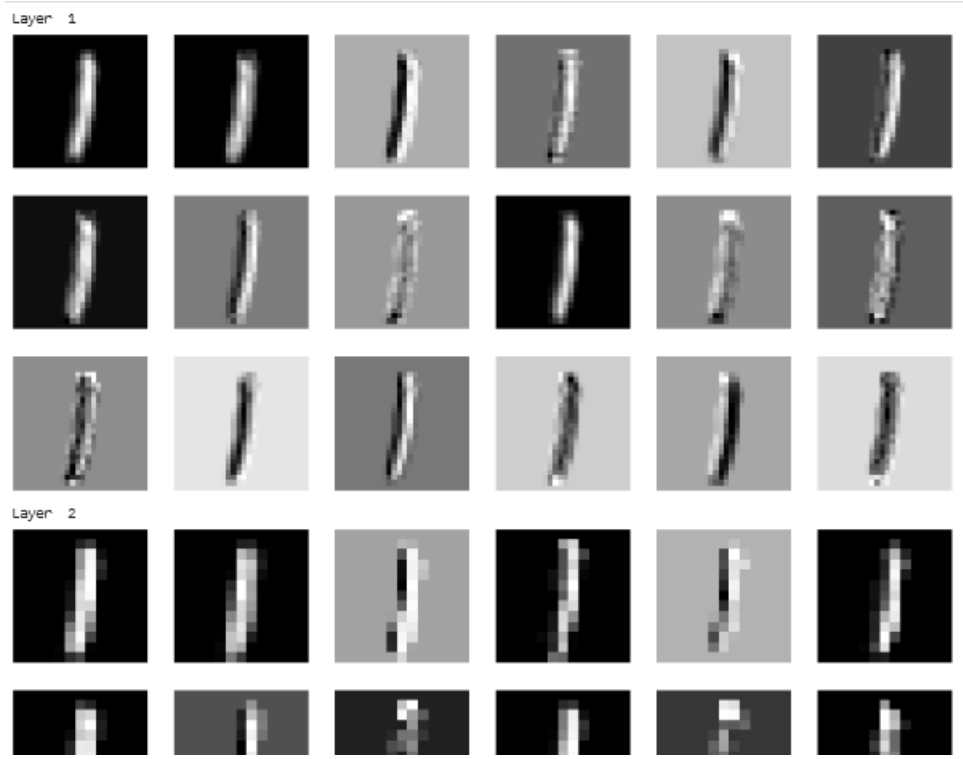
5. Use the model

```
def visualize_featureMaps_partial(outputs,num_imagem):
    for num_layer in range(len(outputs)):
        plt.figure(figsize=(14, 7))
        layer_viz = outputs[num_layer][num_imagem, :, :, :]
        layer_viz = layer_viz.data
        print("Layer ",num_layer+1)
        for i, filter in enumerate(layer_viz):
            if i == 18:
                break
            plt.subplot(3, 6, i + 1)
            plt.imshow(filter.cpu(), cmap='gray')
            plt.axis("off")
        plt.show()
        plt.close()

visualize_featureMaps_partial(outputs,6)
```

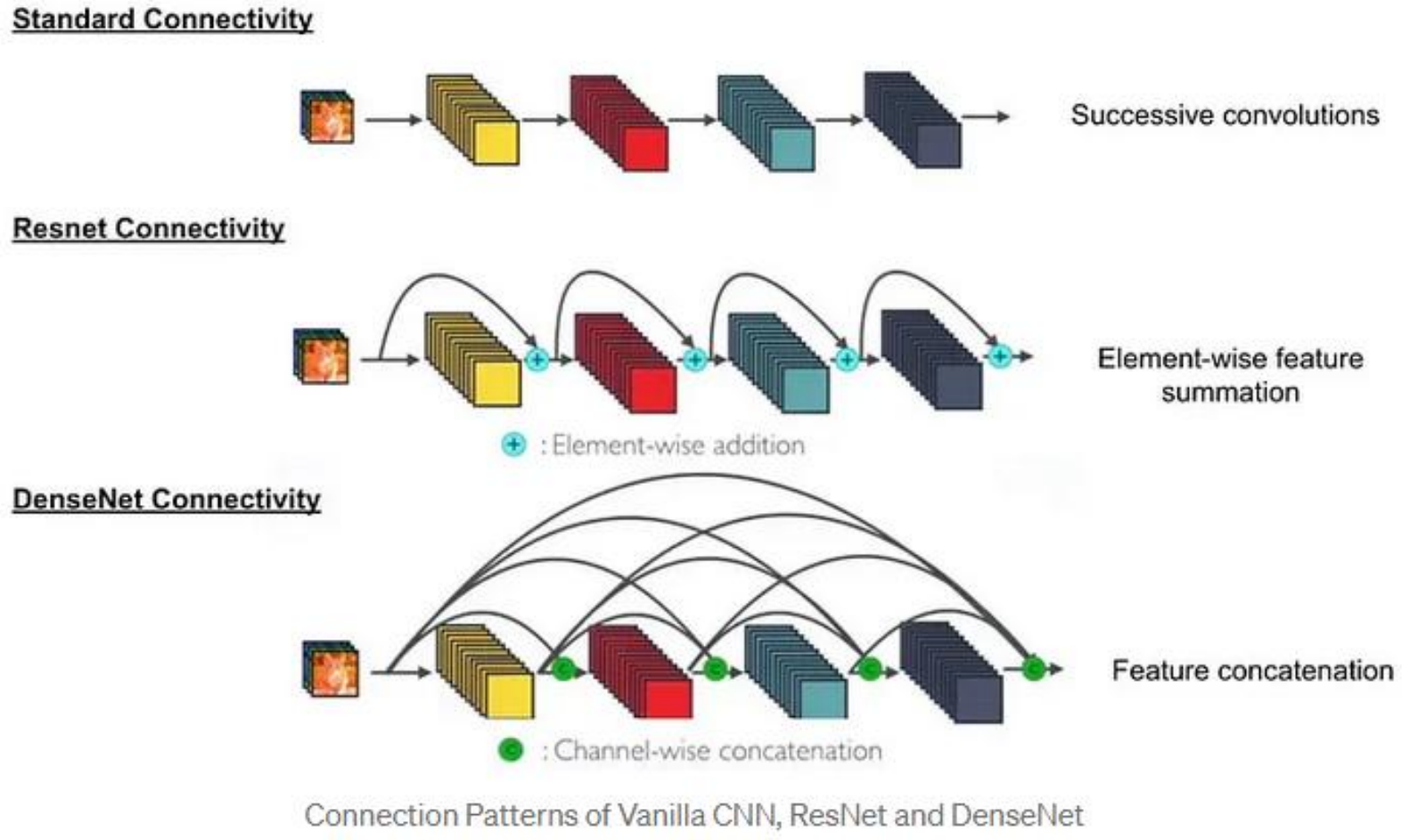
5. Use the model

```
def visualize_featureMaps_partial(outputs,num_imagem):
    for num_layer in range(len(outputs)):
        plt.figure(figsize=(14, 7))
        layer_viz = outputs[num_layer][num_imagem, :, :, :]
        layer_viz = layer_viz.data
        print("Layer ",num_layer+1)
        for i, filter in enumerate(layer_viz):
            if i == 18:
                break
            plt.subplot(3, 6, i + 1)
            plt.imshow(filter.cpu(), cmap='gray')
            plt.axis("off")
        plt.show()
        plt.close()
```

visualize_featureMaps_partial(outputs,6)



Deep neural networks dead weights and units



Connection Patterns of Vanilla CNN, ResNet and DenseNet

https://cvinvolution.medium.com/why-isnt-densenet-adopted-as-extensive-as-resnet-1bee84101160