PUCRS CENTRO DE EDUCAÇÃO CONTINUADA
ESPECIALIZAÇÃO EM CIÊNCIA DE DADOS

LEANDRO ROSSETTI DE SOUZA

Projeto Final da Disciplina Deep Learning I

Porto Alegre
2019

PUCRS CENTRO DE EDUCAÇÃO CONTINUADA
ESPECIALIZAÇÃO EM CIÊNCIA DE DADOS

Trabalho apresentado como requisito para compor a avaliação na disciplina Deep Learning I

Professor: Prof.ºJônatas Wehrmann

Um classificador de imagens baseado em Redes Neurais Convolucionais Profundas utilizando o modelo VGG16, AlexNet e GoogLeNet implementados no Pytorch

A proposta do presente trabalho é especializar, utilizando a técnica de Transfer Learning, um classificador de imagens à partir dos modelos VGG16 e AlexNet pré treinados, tornando-os capazes de classificar imagens de formigas e abelhas não presentes nos datasets originais

Objetivos do trabalho

- Compreender o protocolo completo de treinamento, validação e avaliação de redes profundas e/ou recorrentes.
  - Esse objetivo foi atingido através do processo de implementar o presente classificador, iniciando com a pesquisa de funcionalidades e escolha do modelo, passando pela coleta e tratamento das imagens para compor o dataset de treino e validação e culminando com ao teste de hiperparametros e ajustes necessários no modelo final
- Compreender o impacto de diferentes escolhas de arquiteturas e hiperparâmetros no processo de treinamento.
  - Esse objetivo foi atingido ao analisar o resultado de diversas configurações de hiperparâmetros no processo de ajuste do modelo proposto, esses passos serão explicados ao longo deste trabalho
  -

- Compreender as possibilidades existentes no uso das informações obtidas através de ConvNets.
  - Esse objetivo foi atingido ao pesquisar a técnica de transferência de conhecimento (Transfer Learning) para utilização nesse trabalho, essa técnica abriu um leque de possibilidades a partir da limitação de hardware enfrentada.
- Ter experiência com um framework de Deep Learning
  - Esse objetivo foi atingido ao trabalhar com o pytorch e pesquisar as funcionalidades disponibilizadas pelo mesmo e utilizadas nesse trabalho
- Executar experimentos de acordo com uma metodologia.
  - Esse objetivo foi atingido ao definir a metodologia e escolher o processo de execução utilizado no presente trabalho
- Avaliar os resultados obtidos.
  - Esse objetivo foi atingido analisando e documentodo os resultados obtidos durante o processo de execução desse trabalho

Torchvision Models Implementado nesse trabalho

In [18]:
```
import torch
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as F
from torch import nn
from torchvision import datasets, transforms, models
```

O pacote **models** contém definições de modelos para lidar com diferentes tarefas, incluindo: classificação de imagens, segmentação semântica em pixels, detecção de objetos, segmentação de instância, detecção de ponto-chave da pessoa e classificação de vídeo. Nesse trabalho foram utilizados três dos modelos disponibilizados nesse pacote

O pacote models contém definições para as diversas arquiteturas de modelos

Nesse trabalho foram utilizadas as arquiteturas

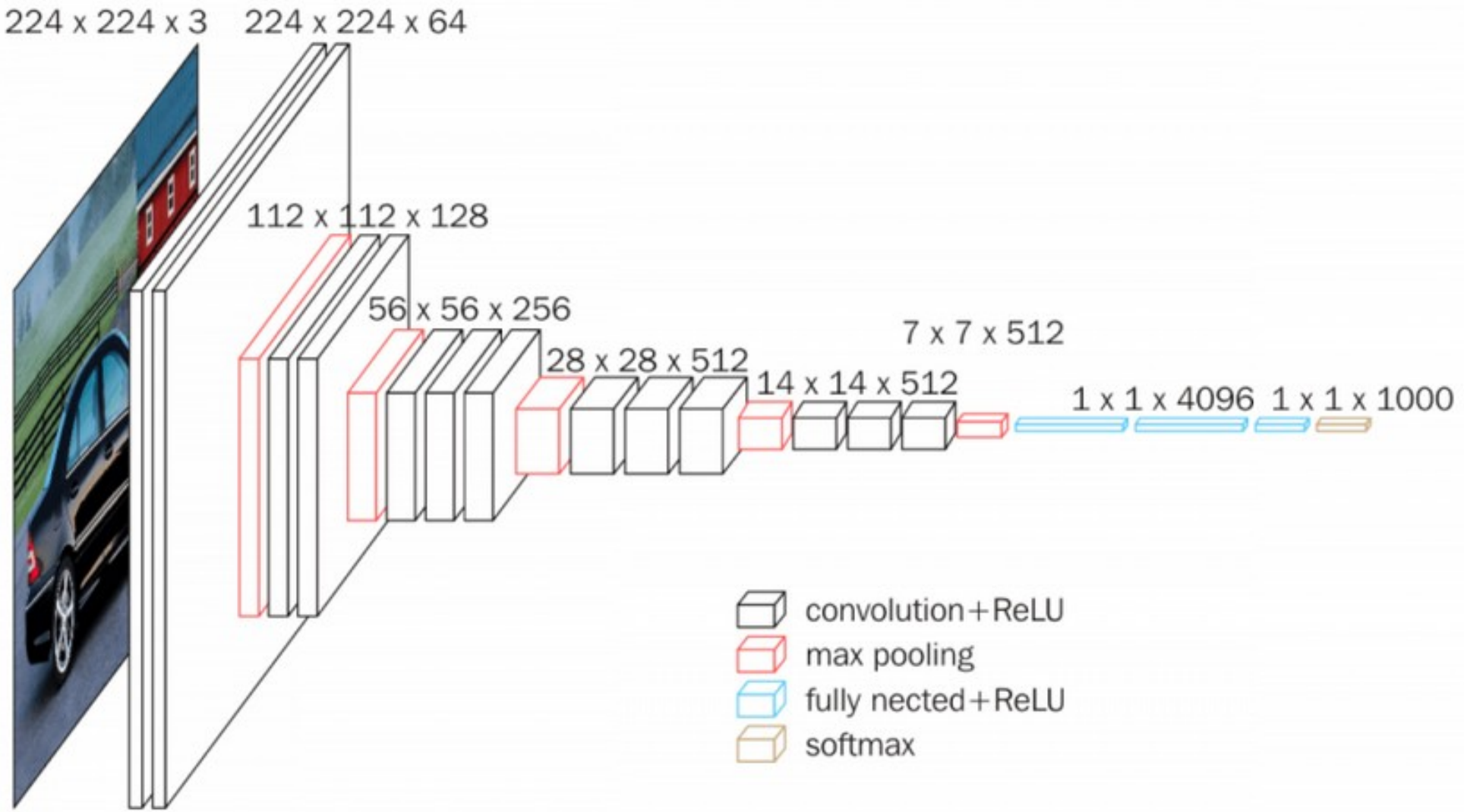
AlexNet

VGG16

O Modelo VVG16

VGG16 é um modelo de rede neural convolucional proposto por K. Simonyan e A. Zisserman, da Universidade de Oxford, no artigo "Redes convolucionais muito profundas para reconhecimento de imagens em larga escala". O modelo alcança 92,7% de precisão no top 5 no ImageNet, que é um conjunto de dados de mais de 14 milhões de imagens pertencentes a 1000 classes. Foi um dos famosos modelos submetidos ao ILSVRC-2014. Ele aprimora o AlexNet substituindo os filtros grandes do kernel-sized (11 e 5 na primeira e segunda camada convolucional, respectivamente) por vários filtros kernel-sized 3 × 3 um após o outro. Mesmo utilizando GPUs NVIDIA Titan Black o VGG16 foi treinado ao longo de semanas

- VGG16 – Convolutional Network for Classification and Detection



Modelo da rede VVG16 implementado nesse trabalho

Na celula abaixo são definidos os modelos que serão utilizados no decorrer do código

In [8]:
```
# Cria mo modelo VGG16
modelVgg = models.vgg16(pretrained=True)
```

Print do modelo VGG criado

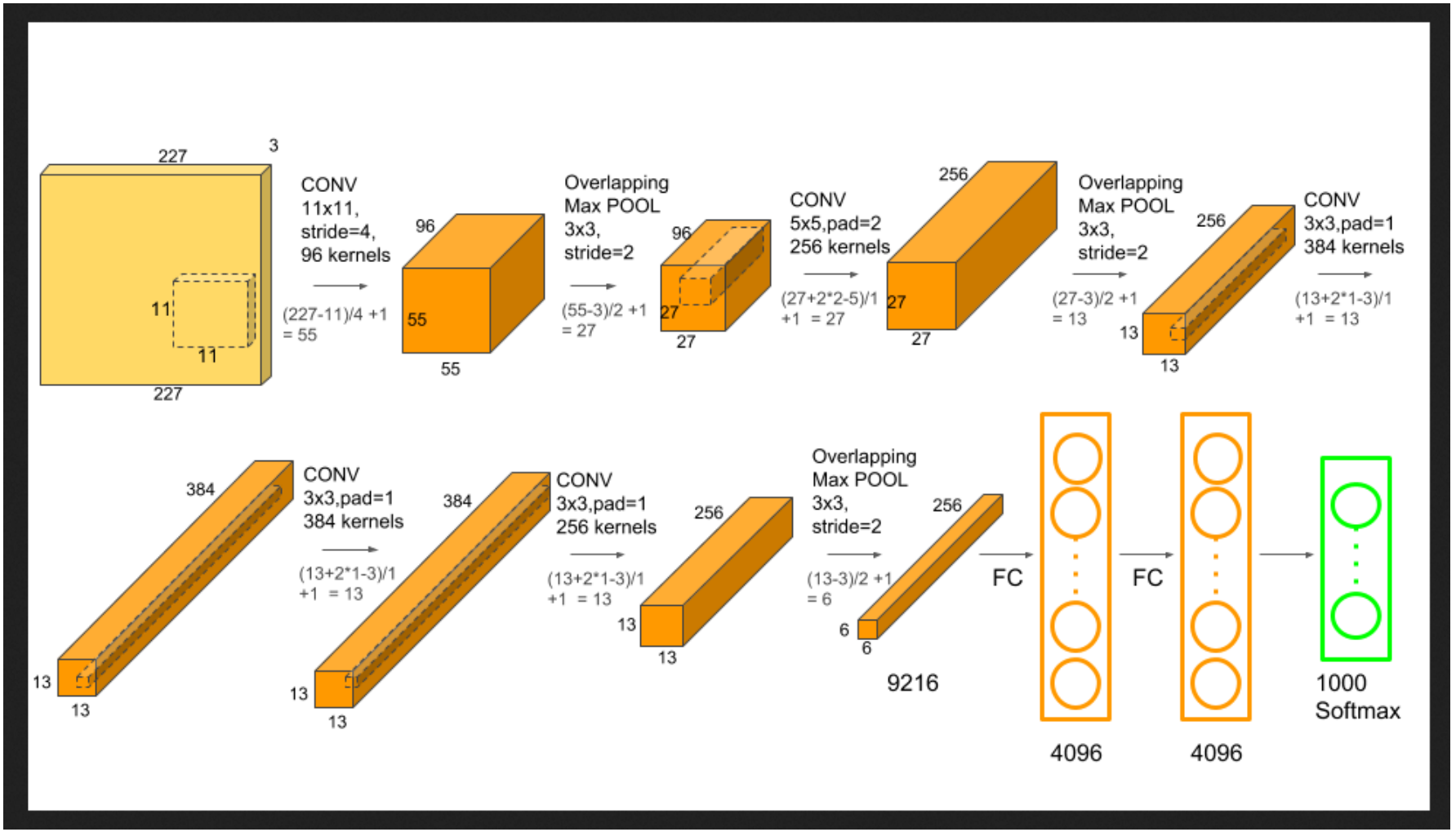
In [9]:
```
print(modelVgg)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

O Modelo AlexNet

O modelo AlexNet era muito maior do que as CNNs anteriores usadas para tarefas de visão computacional (por exemplo, o artigo LeNet de Yann LeCun em 1998). Possui 60 milhões de parâmetros e 650.000 neurônios e levou de cinco a seis dias para treinar em duas GPUs GTX 580 de 3 GB. Hoje, existem CNNs muito mais complexas que podem ser executadas em GPUs mais rápidas com muita eficiência, mesmo em conjuntos de dados muito grandes. Mas em 2012, isso foi grande!

AlexNet Architecture



Modelo da rede AlexNet implementado nesse trabalho

Na celula abaixo são definidos os modelos que serão utilizados no decorrer do código

In [8]:
```
# Cria mo modelo AlexNet
modelAlexNet = models.alexnet(pretrained=True)
```

In [35]:
```
print(modelAlexNet)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
```

Estou utilizando modelos pré-treinados

In [8]:
```
# Cria mo modelo AlexNet
modelAlexNet = models.alexnet(pretrained=True)

# Cria mo modelo VGG16
modelVgg = models.vgg16(pretrained=True)
```

Em consequência disso, na célula abaixo utilizo um loop para "congelar" a parte de extração de features do modelo para não perder o "conhecimento" já acumulado.

In [11]:
```
for param in modelVgg.features.parameters():
  param.requires_grad = False

for param in modelAlexNet.features.parameters():
  param.requires_grad = False
```

Em ambos os modelos a camada de saída possui 1000 classes, foi preciso ajustar o modelo para o novo dataset que possui apenas 2 classes

Saída VGG16

```
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
```

Saída AlexNet

```
  (5): ReLU(inplace=True)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
```

Isso foi feito com o código abaixo:

```
#Reduz o numero de classes de saida de 1000 para 2 no modelo VGG
n_inputs = modelVgg.classifier[6].in_features
last_layer = nn.Linear(n_inputs, len(classes))
modelVgg.classifier[6] = last_layer
modelVgg.to(device)
print(modelVgg.classifier)

#Reduz o numero de classes de saida de 1000 para 2 no modelo AlexNet
n_inputs = modelAlexNet.classifier[6].in_features
last_layer = nn.Linear(n_inputs, len(classes))
modelAlexNet.classifier[6] = last_layer
modelAlexNet.to(device)
print(modelAlexNet.classifier)
```

```
Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=2, bias=True)
)
Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Linear(in_features=9216, out_features=4096, bias=True)
  (2): ReLU(inplace=True)
  (3): Dropout(p=0.5, inplace=False)
  (4): Linear(in_features=4096, out_features=4096, bias=True)
  (5): ReLU(inplace=True)
  (6): Linear(in_features=4096, out_features=2, bias=True)
)
```

Foram definidos para teste nesse trabalho duas funções de custo [CrossEntropy, NLLLoss] e dois otimizadores [Adam, SGD], definidos abaixo:

In [13]:
```
# Definição extraida da documentação disponível em: https://pytorch.org/docs/stable/nn.html#
# This criterion combines nn.LogSoftmax() and nn.NLLLoss() in one single class.
# It is useful when training a classification problem with C classes.
criterionCrossEntropy = nn.CrossEntropyLoss()

# Definição extraida da documentação disponível em: https://pytorch.org/docs/stable/nn.html#
# The negative log likelihood loss. It is useful to train a classification problem with C classes.
criterionNLLL = nn.NLLLoss()

# Definição extraida da documentação disponível em: https://pytorch.org/docs/stable/optim.html#
# Implements Adam algorithm.
optimizerAdamVgg = torch.optim.Adam(modelVgg.parameters(), lr = 0.0001)
optimizerAdamAlexNet = torch.optim.Adam(modelAlexNet.parameters(), lr = 0.0001)

# Definição extraida da documentação disponível em: https://pytorch.org/docs/stable/optim.html#
# O optimizer SGD Implements stochastic gradient descent (optionally with momentum)
optimizerSGDVgg = torch.optim.SGD(modelVgg.parameters(), lr = 0.0001)
optimizerSGDAlexNet = torch.optim.SGD(modelAlexNet.parameters(), lr = 0.0001)
```

O conjunto de testes será composto por dois modelos e quatro duplas de Otimizadores e Funções de custo detalhadas a seguir:

- **Para o modelo VGG16**
  1. Função de custo CrossEntropy e Otimizador Adam
  2. Função de custo CrossEntropy e Otimizador SGD
  3. Função de custo NLLL e Otimizador Adam
  4. Função de custo NLLL e Otimizador SGD

- **Para o modelo AlexNet**
  1. Função de custo CrossEntropy e Otimizador Adam
  2. Função de custo CrossEntropy e Otimizador SGD
  3. Função de custo NLLL e Otimizador Adam
  4. Função de custo NLLL e Otimizador SGD

Ao todo serão executados oito rodadas de testes cobrindo todos os modelos e hyperparametros definidos acima

Testes do modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam

Treinamento em cinco épocas e validação do modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam

Para testar o modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam execute a celula abaixo

```
# Para o modelo VGG16
# Função de custo CrossEntropy e Otimizador Adam
criterion = criterionCrossEntropy
optimizer = optimizerAdamVgg
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelVgg, epochs)
```

```
epoch : 1
training loss: 0.0276, acc 0.7459
validation loss: 0.0144, validation acc 0.8889
epoch : 2
training loss: 0.0240, acc 0.7828
validation loss: 0.0130, validation acc 0.9412
epoch : 3
training loss: 0.0155, acc 0.8484
validation loss: 0.0158, validation acc 0.8889
epoch : 4
training loss: 0.0111, acc 0.8975
validation loss: 0.0188, validation acc 0.8954
epoch : 5
training loss: 0.0137, acc 0.8934
validation loss: 0.0192, validation acc 0.9020
```

Gráfico comparativo dos valores de custo do treinamento e validação do modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam

Grafico comparativo dos valores de custo do treinamento e validação

```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1711b1f2c50>
```

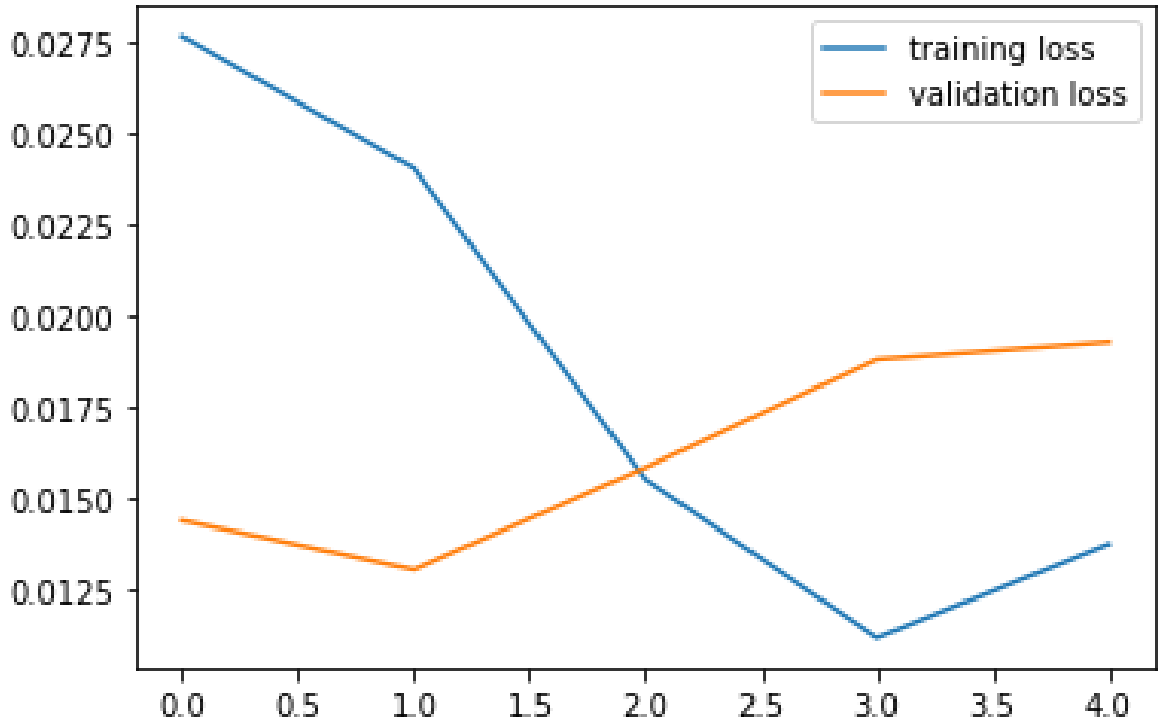
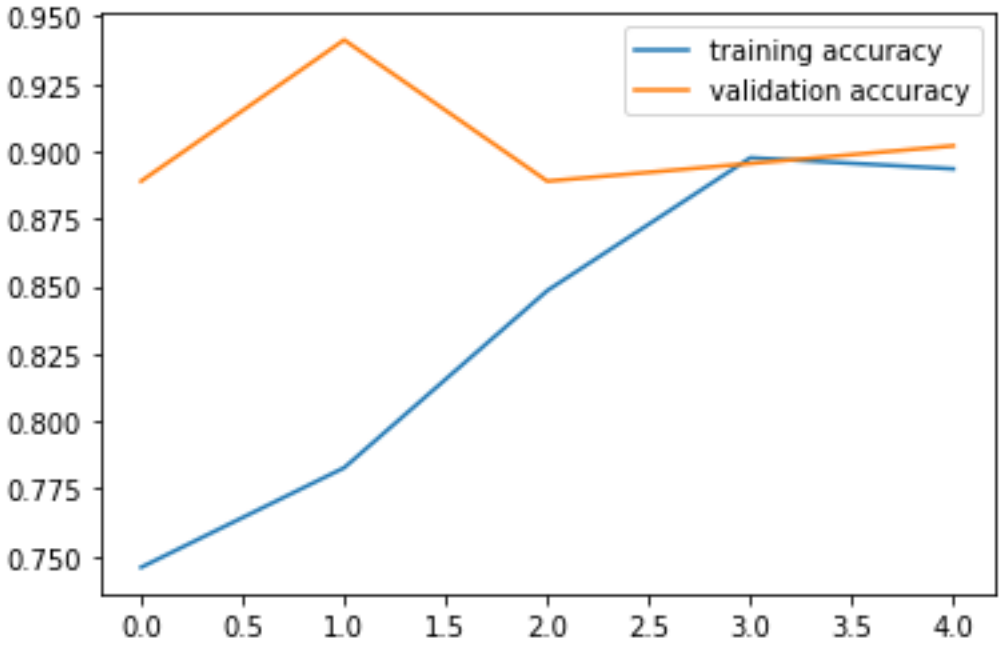


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam

Grafico comparativo dos valores de acuracia do treinamento e validação

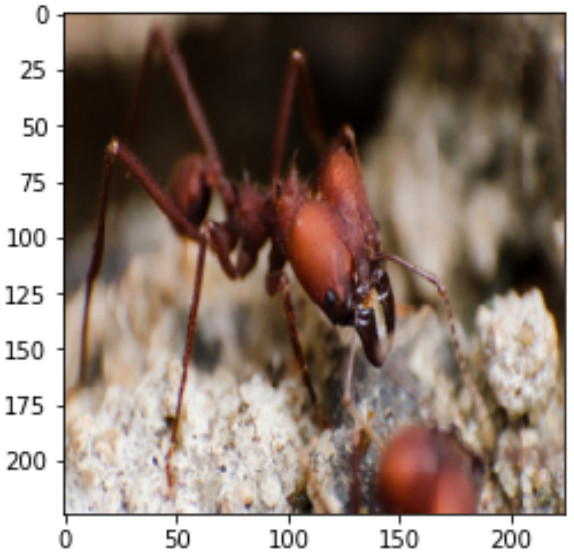
In [17]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

Out[17]: <matplotlib.legend.Legend at 0x1711b226940>



Resultado da validação do modelo VGG com Função de custo CrossEntropy e Otimizador Adam com uma imagem da internet

Out[20]: <matplotlib.image.AxesImage at 0x1713b2d8c18>



Execute o código da celula abaixo para classificar com o modelo VGG a imagem tratada na celula anterior

In [21]:
```
image = img.to(device).unsqueeze(0)
output = modelVgg(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
formiga
```

Batch de 20 imagens de validação submetido ao modelo VGG treinado anteriormente com Função de custo CrossEntropy e Otimizador Adam

Na celula abaixo é submetido um Batch de 20 imagens ao modelo VGG treinado aneriormente

In [22]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelVgg(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25,4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

Podemos conferir na imagem que o modelo acertou aproximadamente 95% das vezes

Testes do modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD

Treinamento em cinco épocas e validação do modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD

Para testar o modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD execute a celula abaixo

In [23]:
```
# Para o modelo VGG16
# Função de custo CrossEntropy e Otimizador SGD
criterion = criterionCrossEntropy
optimizer = optimizerSGDVgg
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelVgg, epochs)
```

```
epoch : 1
training loss: 0.0090, acc 0.9139
validation loss: 0.0187, validation acc 0.9085
epoch : 2
training loss: 0.0105, acc 0.9016
validation loss: 0.0163, validation acc 0.9216
epoch : 3
training loss: 0.0088, acc 0.9221
validation loss: 0.0180, validation acc 0.9150
epoch : 4
training loss: 0.0127, acc 0.8648
validation loss: 0.0176, validation acc 0.9346
epoch : 5
training loss: 0.0148, acc 0.8607
validation loss: 0.0169, validation acc 0.9346
```

Gráfico comparativo dos valores de custo do treinamento e validação do modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD

Grafico comparativo dos valores de custo do treinamento e validação

In [24]:
```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

Out[24]: <matplotlib.legend.Legend at 0x1713bf76da0>

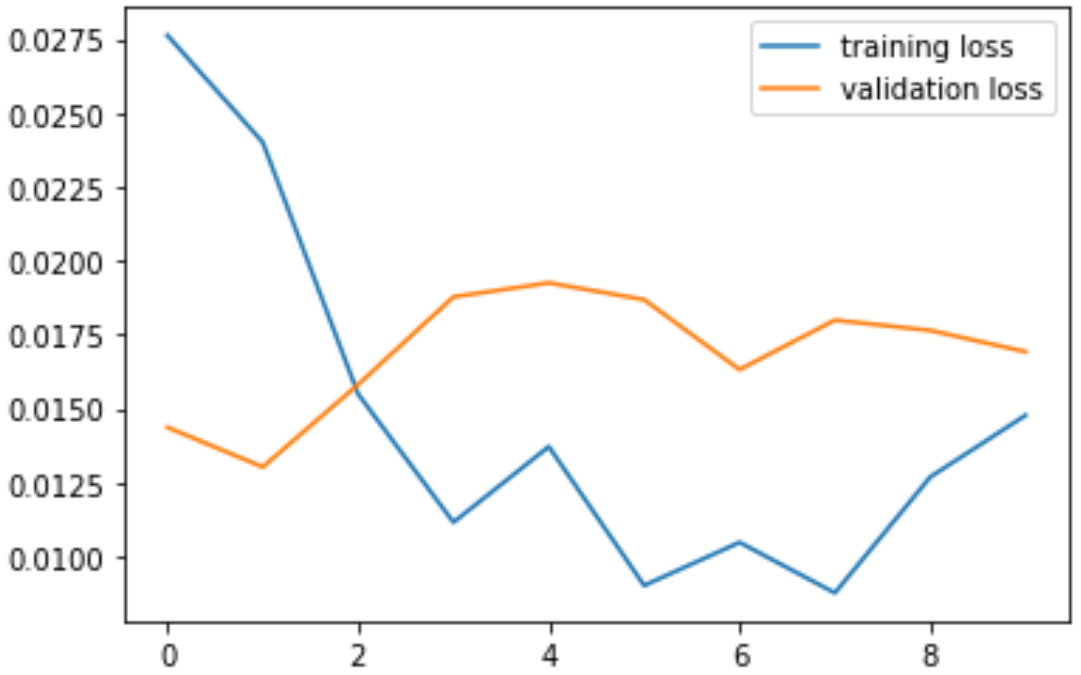


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD

Grafico comparativo dos valores de acuracia do treinamento e validação

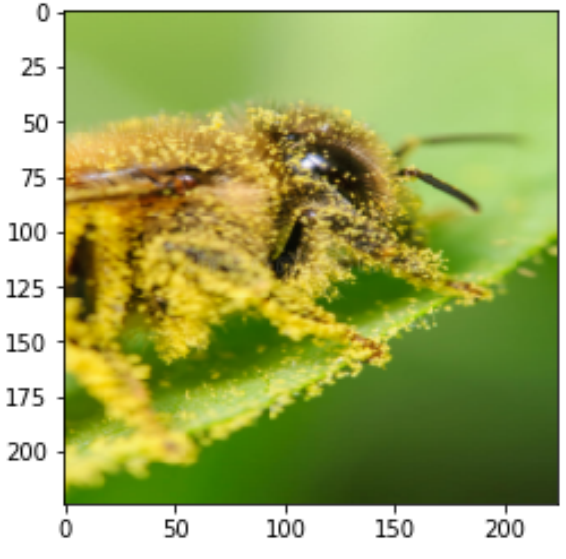
In [25]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x1713ba30198>

Resultado da validação do modelo VGG com Função de custo CrossEntropy e Otimizador SGD com uma imagem da internet

In [31]:
```
img = transform_validation(img)
plt.imshow(im_convert(img))
```

Out[31]:
```
<matplotlib.image.AxesImage at 0x17119710978>
```



Execute o código da celula abaixo para classificar com o modelo VGG a imagem tratada na celula anterior

In [32]:
```
image = img.to(device).unsqueeze(0)
output = modelVgg(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
abelha
```

Batch de 20 imagens de validação submetido ao modelo VGG treinado anteriormente com Função de custo CrossEntropy e Otimizador SGD

Na celula abaixo é submetido um Batch de 20 imagens ao modelo VGG treinado aneriormente

```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelVgg(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25,4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

Podemos conferir na imagem que o modelo acertou aproximadamente 90% das vezes

Testes do modelo VGG16 com Função de custo NLLL e Otimizador Adam

Treinamento em cinco épocas e validação do modelo VGG16 com Função de custo NLLL e Otimizador ADAM

Para testar o modelo VGG16 com Função de custo NLLL e Otimizador Adam execute a celula abaixo

[34]:
```
# Para o modelo VGG16
# Função de custo NLLL e Otimizador Adam
criterion = criterionNLLL
optimizer = optimizerAdamVgg
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelVgg, epochs)
```

```
epoch : 1
training loss: -1.3269, acc 0.8402
validation loss: -6.5844, validation acc 0.8758
epoch : 2
training loss: -19.1292, acc 0.8361
validation loss: -60.9240, validation acc 0.9216
epoch : 3
training loss: -114.6472, acc 0.7705
validation loss: -267.1681, validation acc 0.8366
epoch : 4
training loss: -381.4782, acc 0.6844
validation loss: -787.3440, validation acc 0.6732
epoch : 5
training loss: -967.9043, acc 0.6189
validation loss: -1811.6963, validation acc 0.5621
```

Para testar o modelo VGG16 com Função de custo NLLL e Otimizador Adam execute a celula abaixo

```
# Para o modelo VGG16
# Função de custo NLLL e Otimizador Adam
criterion = criterionNLLL
optimizer = optimizerAdamVgg
epochs = 10

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelVgg, epochs)
```

```
epoch : 1
training loss: -1842.8679, acc 0.5287
validation loss: -3526.9435, validation acc 0.5359
epoch : 2
training loss: -3860.7961, acc 0.5000
validation loss: -6183.1170, validation acc 0.4902
epoch : 3
training loss: -5764.7626, acc 0.4590
validation loss: -10002.5918, validation acc 0.5163
epoch : 4
training loss: -9613.8258, acc 0.5246
validation loss: -14987.2163, validation acc 0.5490
epoch : 5
training loss: -14125.3810, acc 0.5246
validation loss: -21699.9745, validation acc 0.5229
epoch : 6
training loss: -20755.7979, acc 0.5533
validation loss: -30108.0678, validation acc 0.5163
epoch : 7
training loss: -27831.5993, acc 0.5082
validation loss: -40438.0351, validation acc 0.5294
epoch : 8
training loss: -34220.4338, acc 0.5410
validation loss: -52556.5114, validation acc 0.4641
epoch : 9
training loss: -49591.5740, acc 0.5123
validation loss: -66797.8897, validation acc 0.5163
epoch : 10
training loss: -58493.7041, acc 0.5164
validation loss: -83572.0678, validation acc 0.5033
```

Como o resultado em 5 épocas foi muito fraco, decidi aumentar para 10 épocas, porem o resultado piorou.

Gráfico comparativo dos valores de custo do treinamento e validação do modelo VGG16 com Função de custo NLLL e Otimizador ADAM

Grafico comparativo dos valores de custo do treinamento e validação

In [36]:
```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

Out[36]:
```
<matplotlib.legend.Legend at 0x1711a8bef60>
```

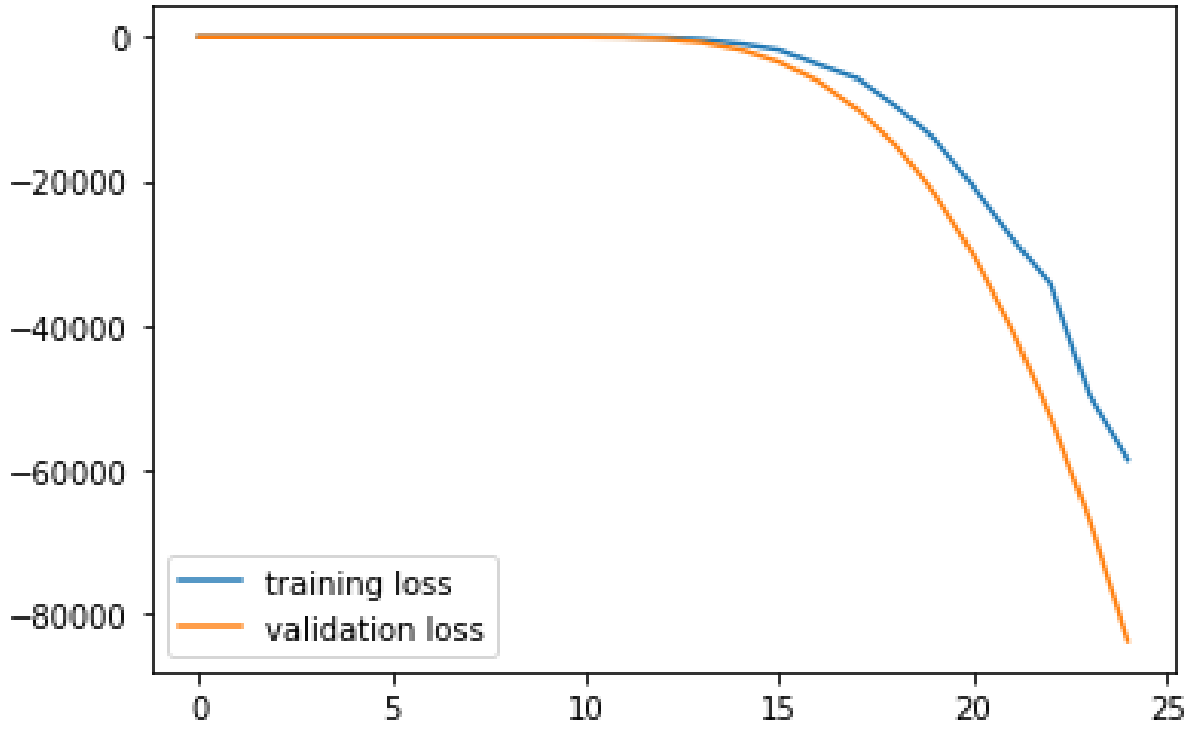
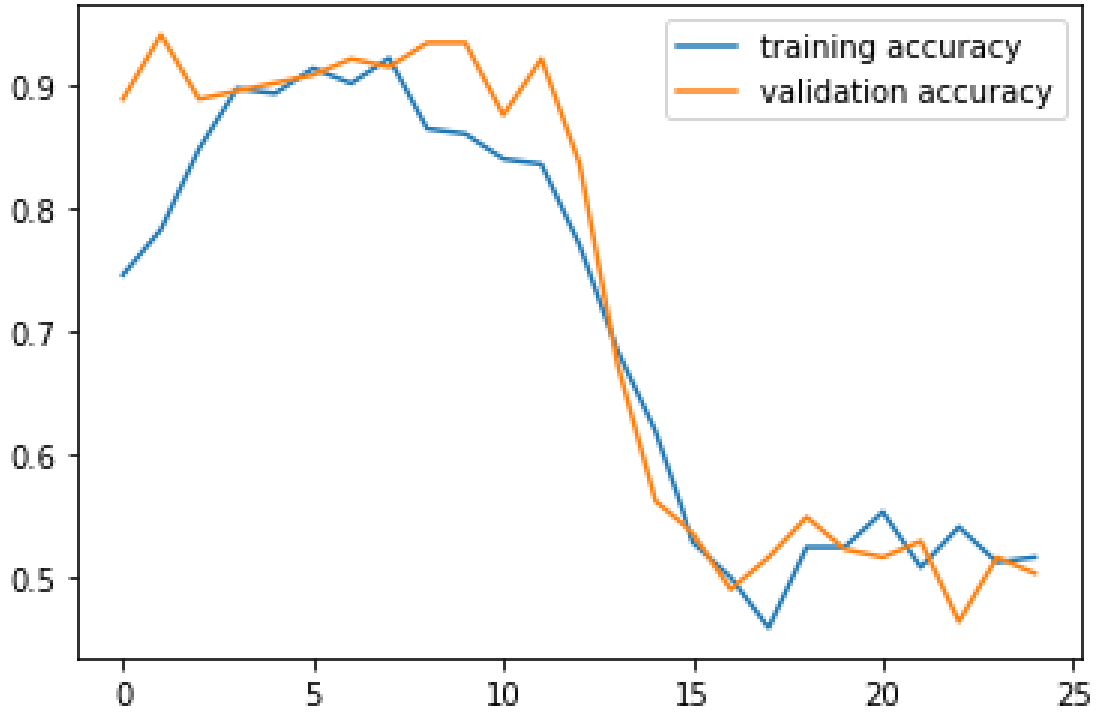


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo VGG16 com Função de custo NLLL e Otimizador ADAM

Grafico comparativo dos valores de acuracia do treinamento e validação

In [37]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

Out[37]:
```
<matplotlib.legend.Legend at 0x1711a31a4e0>
```

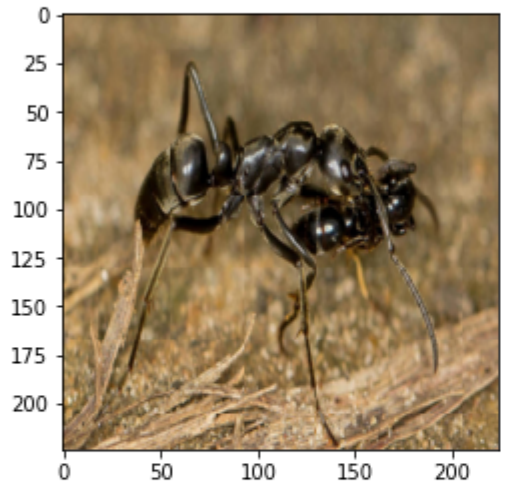


Resultado da validação do modelo VGG com Função de custo NLLL e Otimizador ADAM com uma imagem da internet

Na celula abaixo é feito o tratamento da imagem carregada anteriorente para deixa-la pronta para ser submetida ao classificador

In [39]:
```
img = transform_validation(img)
plt.imshow(im_convert(img))
```

Out[39]:
```
<matplotlib.image.AxesImage at 0x1713bec5c50>
```



Execute o código da celula abaixo para classificar com o modelo VGG a imagem tratada na celula anterior

In [40]:
```
image = img.to(device).unsqueeze(0)
output = modelVgg(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
formiga
```

Batch de 20 imagens de validação submetido ao modelo VGG treinado anteriormente com Função de custo NLLL e Otimizador ADAM

Na celula abaixo é submetido um Batch de 20 imagens ao modelo VGG treinado aneriormente

In [41]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelVgg(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25,4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

Podemos conferir na imagem que o modelo acertou aproximadamente 25% das vezes

Testes do modelo VGG16 com Função de custo NLLL e Otimizador SGD

Treinamento em cinco épocas e validação do modelo VGG16 com Função de custo NLLL e Otimizador SGD

Para testar o modelo VGG16 com Função de custo NLLL e Otimizador SGD execute a celula abaixo

In [42]:
```
# Para o modelo VGG16
# Função de custo NLLL e Otimizador SGD
criterion = criterionNLLL
optimizer = optimizerSGDVgg
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelVgg, epochs)
```

```
epoch : 1
training loss: nan, acc 0.4754
validation loss: nan, validation acc 0.5425
epoch : 2
training loss: nan, acc 0.4959
validation loss: nan, validation acc 0.5425
epoch : 3
training loss: nan, acc 0.4959
validation loss: nan, validation acc 0.5425
epoch : 4
training loss: nan, acc 0.4959
validation loss: nan, validation acc 0.5425
epoch : 5
training loss: nan, acc 0.4959
validation loss: nan, validation acc 0.5425
```

Gráfico comparativo dos valores de custo do treinamento e validação do modelo VGG16 com Função de custo NLLL e Otimizador DGD

Grafico comparativo dos valores de custo do treinamento e validação

In [43]:
```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

Out[43]: <matplotlib.legend.Legend at 0x17116eaaf28>

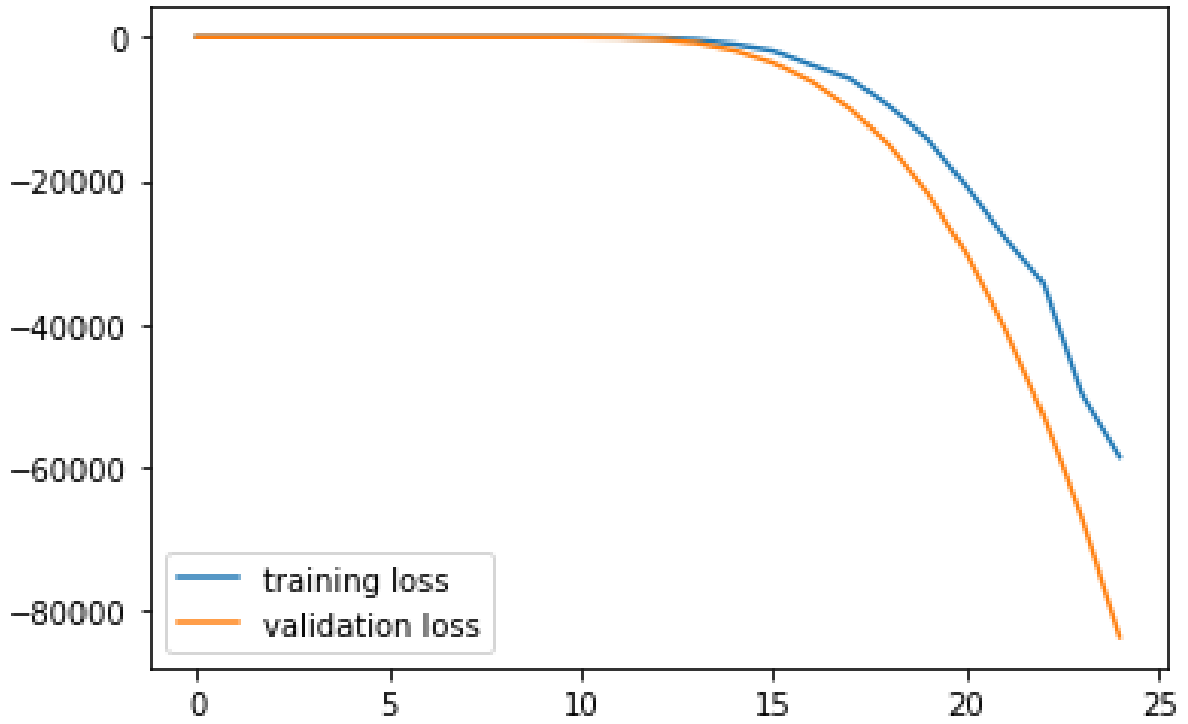
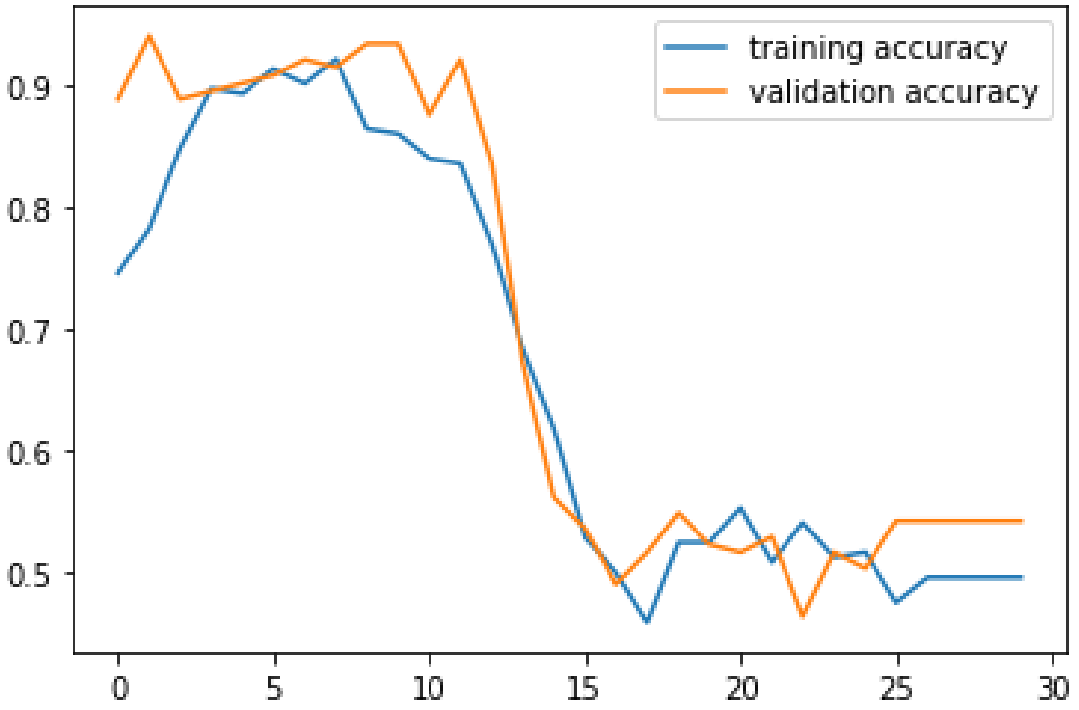


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo VGG16 com Função de custo NLLL e Otimizador SGD

Grafico comparativo dos valores de acuracia do treinamento e validação

In [44]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

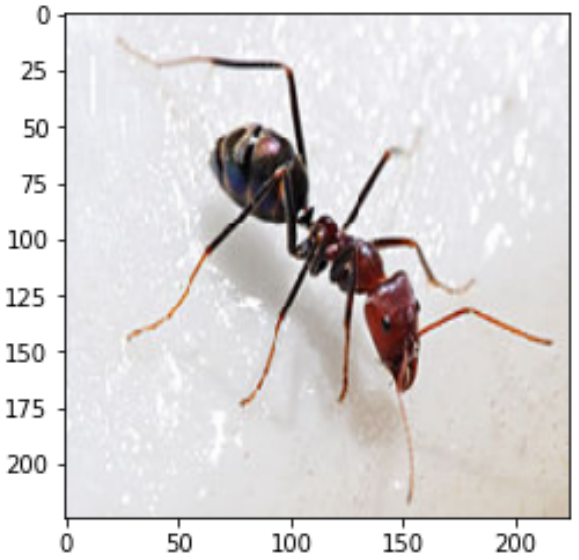
Out[44]:
```
<matplotlib.legend.Legend at 0x17116f1b9b0>
```



Resultado da validação do modelo VGG com Função de custo NLLL e Otimizador SGD com uma imagem da internet

In [46]:
```
img = transform_validation(img)
plt.imshow(im_convert(img))
```

Out[46]:
```
<matplotlib.image.AxesImage at 0x17116eb0b70>
```



Execute o código da celula abaixo para classificar com o modelo VGG a imagem tratada na celula anterior

In [47]:
```
image = img.to(device).unsqueeze(0)
output = modelVgg(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
abelha
```

Batch de 20 imagens de validação submetido ao modelo VGG treinado anteriormente com Função de custo NLLL e Otimizador SGD

Na celula abaixo é submetido um Batch de 20 imagens ao modelo VGG treinado aneriormente

In [56]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelVgg(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25,4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

Podemos conferir na imagem que o modelo acertou aproximadamente 50% das vezes

Testes do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam

Treinamento em cinco épocas e validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam

Para testar o modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam execute a celula abaixo

In [57]:
```
# Para o modelo AlexNet
# Função de custo CrossEntropy e Otimizador Adam
criterion = criterionCrossEntropy
optimizer = optimizerAdamAlexNet
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: 0.0356, acc 0.5820
validation loss: 0.0194, validation acc 0.8497
epoch : 2
training loss: 0.0237, acc 0.7705
validation loss: 0.0178, validation acc 0.8824
epoch : 3
training loss: 0.0230, acc 0.7623
validation loss: 0.0158, validation acc 0.8889
epoch : 4
training loss: 0.0215, acc 0.7828
validation loss: 0.0169, validation acc 0.8693
epoch : 5
training loss: 0.0206, acc 0.8279
validation loss: 0.0178, validation acc 0.8562
```

In [58]:
```
# Para o modelo AlexNet
# Função de custo CrossEntropy e Otimizador Adam
criterion = criterionCrossEntropy
optimizer = optimizerAdamAlexNet
epochs = 10

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: 0.0225, acc 0.7705
validation loss: 0.0199, validation acc 0.8431
epoch : 2
training loss: 0.0196, acc 0.8156
validation loss: 0.0180, validation acc 0.8693
epoch : 3
training loss: 0.0168, acc 0.8443
validation loss: 0.0222, validation acc 0.8693
epoch : 4
training loss: 0.0167, acc 0.8361
validation loss: 0.0209, validation acc 0.8627
epoch : 5
training loss: 0.0215, acc 0.7869
validation loss: 0.0202, validation acc 0.8889
epoch : 6
training loss: 0.0195, acc 0.8115
validation loss: 0.0175, validation acc 0.8693
epoch : 7
training loss: 0.0141, acc 0.8689
validation loss: 0.0253, validation acc 0.8824
epoch : 8
training loss: 0.0158, acc 0.8361
validation loss: 0.0200, validation acc 0.8758
epoch : 9
training loss: 0.0175, acc 0.8279
validation loss: 0.0221, validation acc 0.8824
epoch : 10
training loss: 0.0131, acc 0.8893
validation loss: 0.0199, validation acc 0.8889
```

Como o resultado em 5 épocas processou rápidamente, decidi aumentar para 10 épocas, porem o resultado não melhorou significativamente.

Gráfico comparativo dos valores de custo do treinamento e validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam

Grafico comparativo dos valores de custo do treinamento e validação

In [59]:
```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

Out[59]:
```
<matplotlib.legend.Legend at 0x1711a1d1208>
```

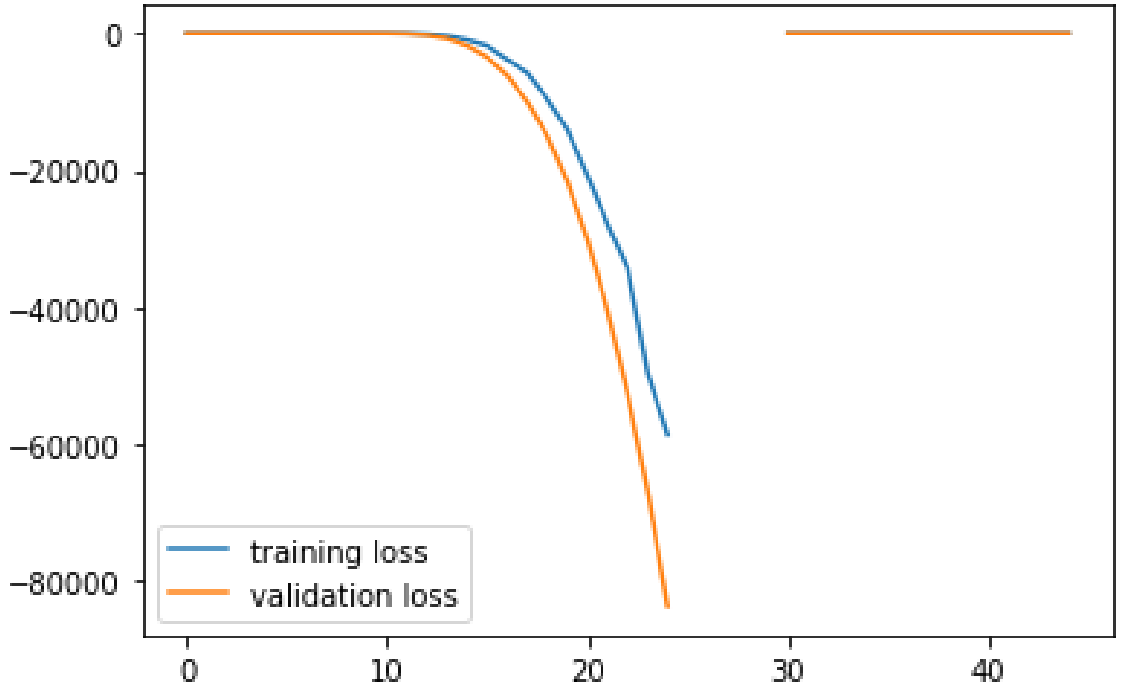


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam

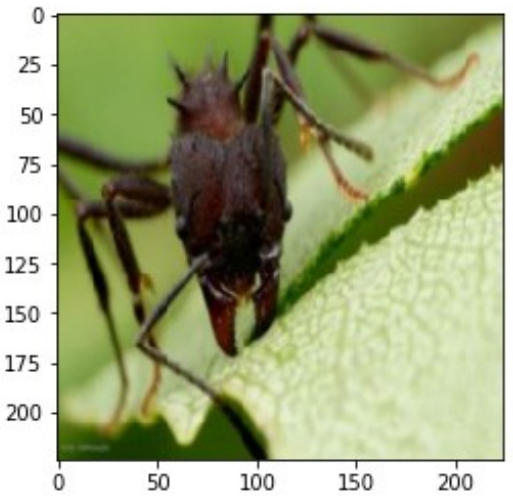
Grafico comparativo dos valores de acuracia do treinamento e validação

In [60]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

Out[60]: <matplotlib.legend.Legend at 0x1711a0b3e48>

Resultado da validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam com uma imagem da internet

Out[62]: <matplotlib.image.AxesImage at 0x1713bf2bc88>



Execute o código da celula abaixo para classificar com o modelo AlexNet a imagem tratada na celula anterior

In [63]:
```
image = img.to(device).unsqueeze(0)
output = modelAlexNet(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
formiga
```

Batch de 20 imagens de validação submetido ao modelo AlexNet treinado anteriormente com Função de custo CrossEntropy e Otimizador Adam

Na celula abaixo é submetido um Batch de 20 imagens ao modelo AlexNet treinado aneriormente

In [64]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelAlexNet(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

Podemos conferir na imagem que o modelo acertou aproximadamente 90% das vezes

Testes do modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD

Treinamento em cinco épocas e validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD

Para testar o modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD execute a celul

In [65]:
```
# Para o modelo AlexNet
# Função de custo CrossEntropy e Otimizador SGD
criterion = criterionNLLL
optimizer = optimizerAdamAlexNet
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: -1.0853, acc 0.8238
validation loss: -5.4982, validation acc 0.8497
epoch : 2
training loss: -27.9436, acc 0.7951
validation loss: -92.5660, validation acc 0.8693
epoch : 3
training loss: -235.2968, acc 0.7008
validation loss: -523.2234, validation acc 0.6340
epoch : 4
training loss: -834.2422, acc 0.5246
validation loss: -1703.0457, validation acc 0.4575
epoch : 5
training loss: -2465.5761, acc 0.5041
validation loss: -4060.2176, validation acc 0.4575
```

Para testar o modelo AlexNet com Função de custo CrossEntropy e Otimizador

In [81]:
```
# Para o modelo AlexNet
# Função de custo CrossEntropy e Otimizador SGD
criterion = criterionNLLL
optimizer = optimizerAdamAlexNet
epochs = 10

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: -0.4576, acc 0.5984
validation loss: -1.5805, validation acc 0.7582
epoch : 2
training loss: -4.4152, acc 0.6393
validation loss: -10.6777, validation acc 0.6732
epoch : 3
training loss: -19.9901, acc 0.5123
validation loss: -43.6931, validation acc 0.5490
epoch : 4
training loss: -69.1810, acc 0.5041
validation loss: -129.7856, validation acc 0.5425
epoch : 5
training loss: -187.2489, acc 0.4959
validation loss: -323.5573, validation acc 0.5425
epoch : 6
training loss: -423.1937, acc 0.4959
validation loss: -692.5827, validation acc 0.5425
epoch : 7
training loss: -859.0101, acc 0.4959
validation loss: -1320.5600, validation acc 0.5425
epoch : 8
training loss: -1511.2806, acc 0.4959
validation loss: -2290.0294, validation acc 0.5425
epoch : 9
training loss: -2684.8587, acc 0.4959
validation loss: -3723.0585, validation acc 0.5425
epoch : 10
training loss: -4038.9029, acc 0.4959
validation loss: -5676.1721, validation acc 0.5425
```

Como o resultado em 5 épocas processou rápidamente, decidi aumentar para 10 épocas, porem o resultado não melhorou significativamente, E assim como no modelo VGG16 o desempenho piorou ao longo do treino.

Gráfico comparativo dos valores de custo do treinamento e validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD

Grafico comparativo dos valores de custo do treinamento e validação

```
In [82]: plt.plot(running_loss_history, label='training loss')
         plt.plot(val_running_loss_history, label='validation loss')
         plt.legend()
```

```
Out[82]: <matplotlib.legend.Legend at 0x17119ce0b00>
```

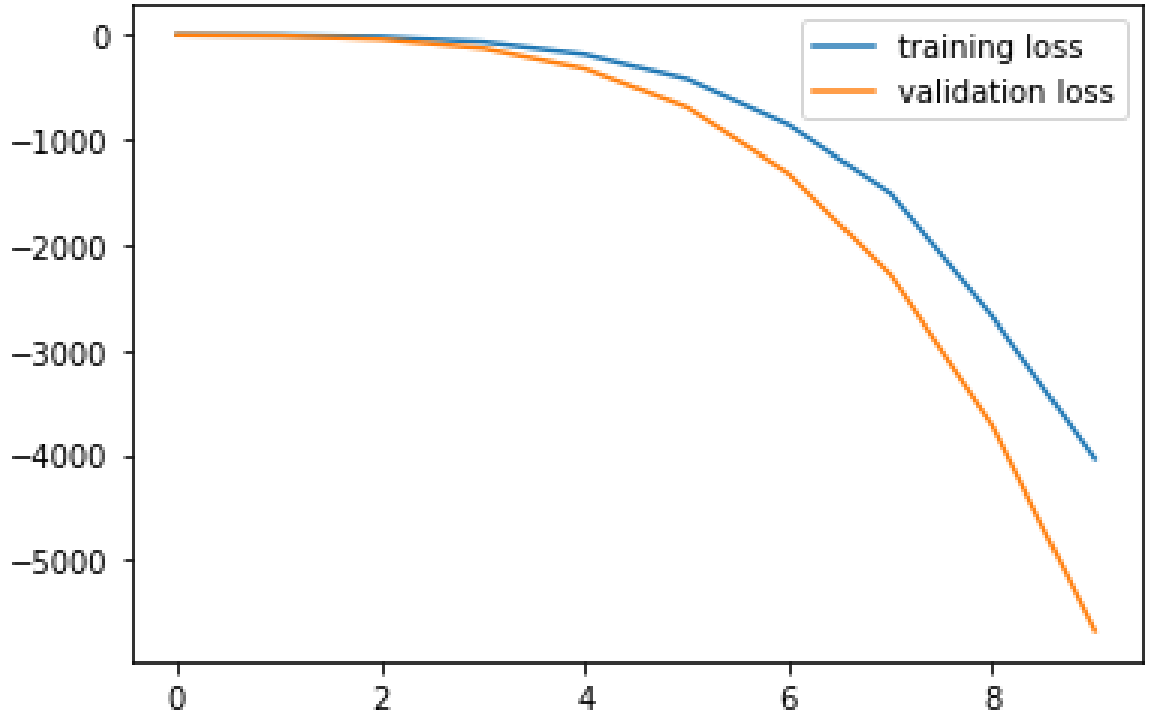
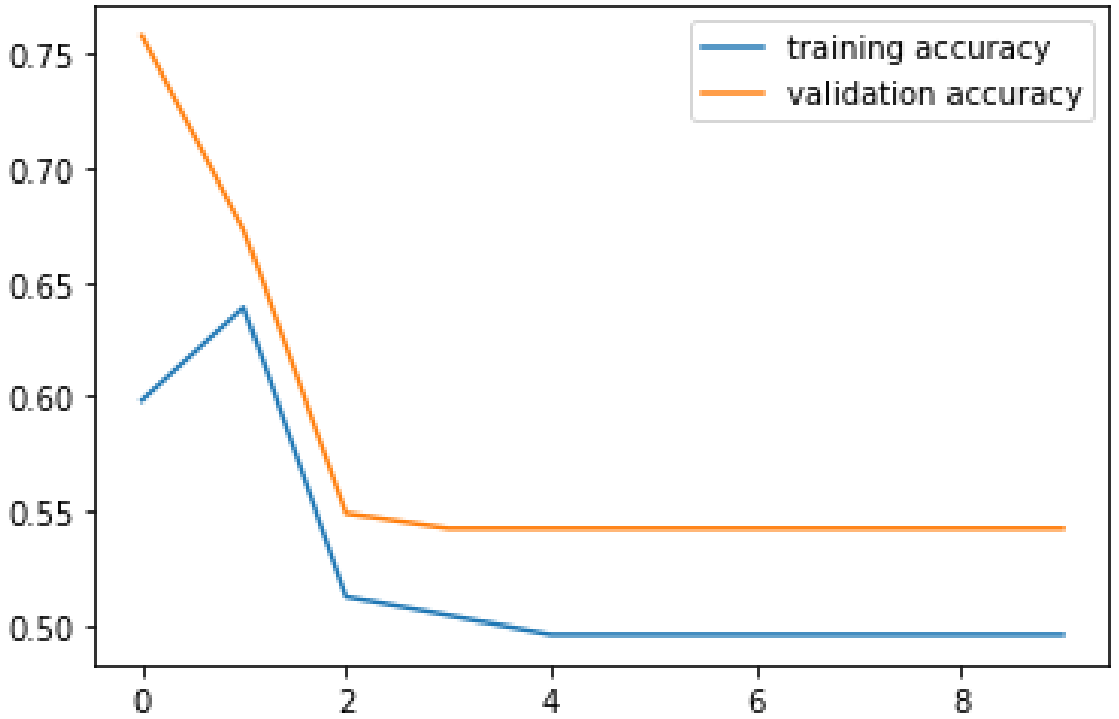


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD

Grafico comparativo dos valores de acuracia do treinamento e validação

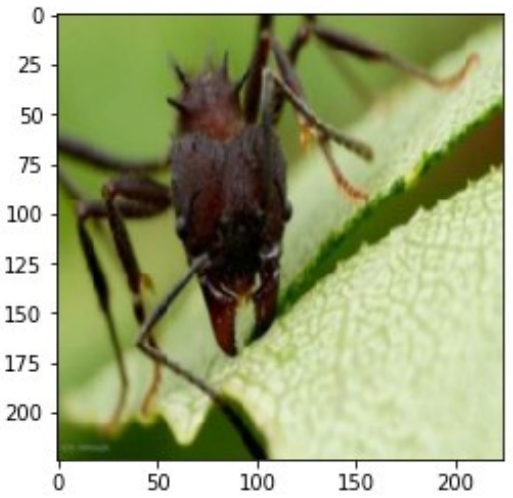
In [83]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

Out[83]:
```
<matplotlib.legend.Legend at 0x17119c7dd68>
```



Resultado da validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD com uma imagem da internet

Out[62]: <matplotlib.image.AxesImage at 0x1713bf2bc88>



Execute o código da celula abaixo para classificar com o modelo AlexNet a imagem tratada na celula anterior

In [63]:
```
image = img.to(device).unsqueeze(0)
output = modelAlexNet(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
formiga
```

Batch de 20 imagens de validação submetido ao modelo AlexNet treinado anteriormente com Função de custo CrossEntropy e Otimizador SGD

Na celula abaixo é submetido um Batch de 20 imagens ao modelo AlexNet treinado aneriormente

In [64]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelAlexNet(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

Podemos conferir na imagem que o modelo acertou aproximadamente 90% das vezes

Testes do modelo AlexNet com Função de custo NLLL e Otimizador Adam

Treinamento em cinco épocas e validação do modelo AlexNet com Função de custo NLLL e Otimizador ADAM

Para testar o modelo AlexNet com Função de custo NLLL e Otimizador Adam (

In [84]:
```
# Para o modelo AlexNet
# Função de custo NLLL e Otimizador Adam
criterion = criterionCrossEntropy
optimizer = optimizerSGDAlexNet
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: 7238.9866, acc 0.5205
validation loss: 5785.0170, validation acc 0.5425
epoch : 2
training loss: 4716.0012, acc 0.4713
validation loss: 5251.6921, validation acc 0.5425
epoch : 3
training loss: 3596.1611, acc 0.4139
validation loss: 650.8753, validation acc 0.5425
epoch : 4
training loss: 2264.2893, acc 0.4959
validation loss: 2868.1346, validation acc 0.4575
epoch : 5
training loss: 1386.0882, acc 0.4877
validation loss: 696.5730, validation acc 0.5425
```

Para testar o modelo AlexNet com Função de custo NLLL e Otimizador Adam

In [85]:
```
# Para o modelo AlexNet
# Função de custo NLLL e Otimizador Adam
criterion = criterionCrossEntropy
optimizer = optimizerSGDAlexNet
epochs = 10

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: 1198.5599, acc 0.4877
validation loss: 634.9528, validation acc 0.4575
epoch : 2
training loss: 925.7609, acc 0.5205
validation loss: 184.7941, validation acc 0.5425
epoch : 3
training loss: 567.7679, acc 0.5287
validation loss: 802.8160, validation acc 0.5425
epoch : 4
training loss: 570.1260, acc 0.5123
validation loss: 232.9087, validation acc 0.5425
epoch : 5
training loss: 400.2725, acc 0.5451
validation loss: 113.3197, validation acc 0.4575
epoch : 6
training loss: 603.1219, acc 0.4385
validation loss: 309.4704, validation acc 0.4575
epoch : 7
training loss: 320.4783, acc 0.4959
validation loss: 1307.1179, validation acc 0.5425
epoch : 8
training loss: 428.1523, acc 0.4795
validation loss: 286.6837, validation acc 0.5425
epoch : 9
training loss: 249.9690, acc 0.5369
validation loss: 205.0976, validation acc 0.4575
epoch : 10
training loss: 285.7392, acc 0.4672
validation loss: 1150.4700, validation acc 0.4575
```

Como o resultado em 5 épocas processou rapidamente, decidi aumentar para 10 épocas, porem o resultado não melhorou significativamente, E assim como no modelo VGG16 o desempenho piorou ao longo do treino.

Gráfico comparativo dos valores de custo do treinamento e validação do modelo AlexNet com Função de custo NLLL e Otimizador ADAM

Grafico comparativo dos valores de custo do treinamento e validação

```
[86]: plt.plot(running_loss_history, label='training loss')
      plt.plot(val_running_loss_history, label='validation loss')
      plt.legend()
```

```
:[86]: <matplotlib.legend.Legend at 0x17116fd38d0>
```

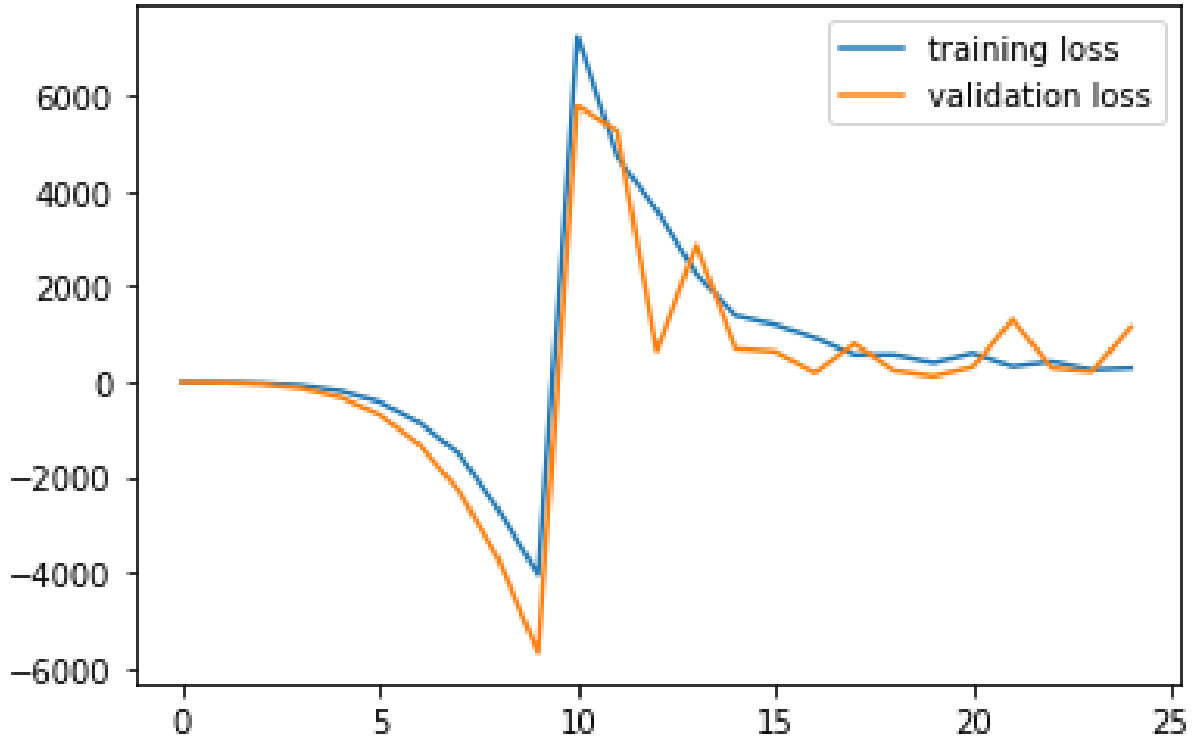
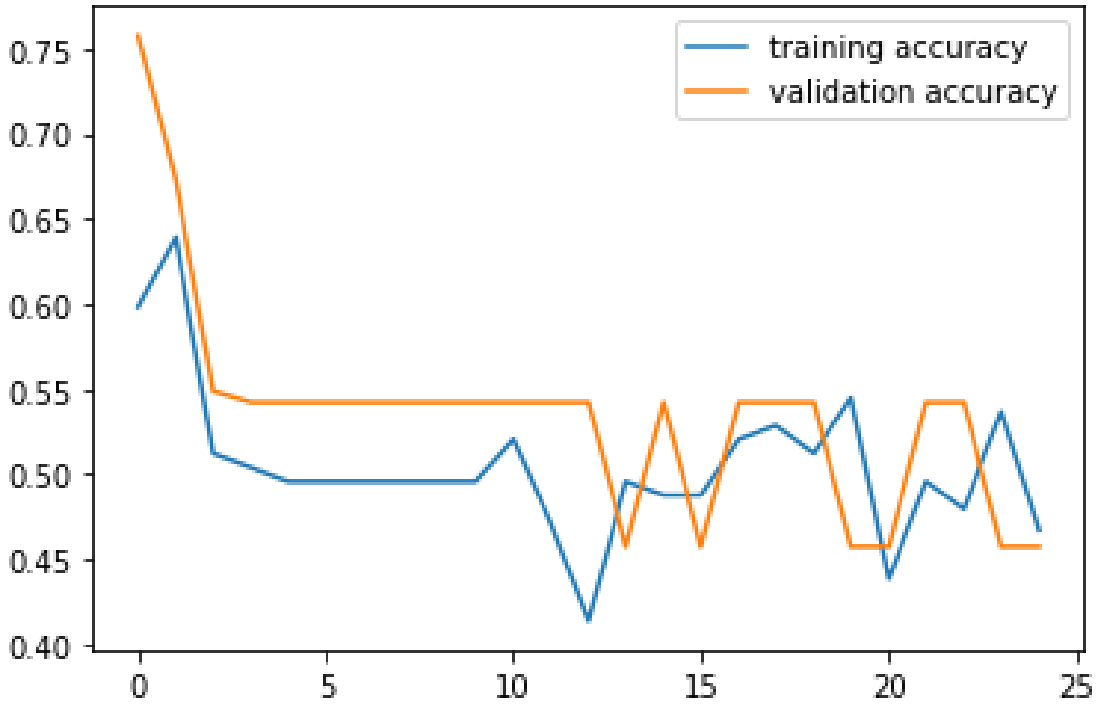


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo AlexNet com Função de custo NLLL e Otimizador ADAM

Grafico comparativo dos valores de acuracia do treinamento e validação

In [87]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

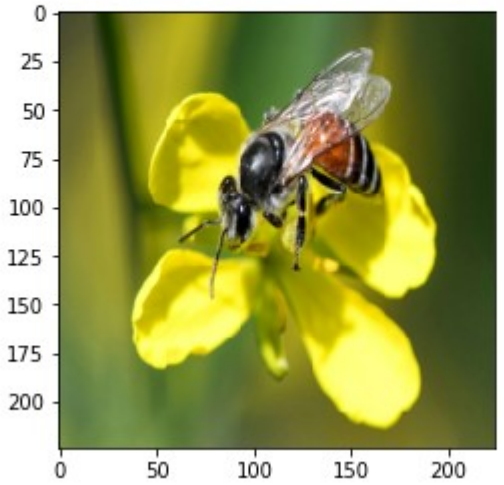
Out[87]:
```
<matplotlib.legend.Legend at 0x1711a85dac8>
```



Resultado da validação do modelo AlexNet com Função de custo NLLL e Otimizador ADAM com uma imagem da internet

```
In [91]: img = transform_validation(img)
         plt.imshow(im_convert(img))
```

```
Out[91]: <matplotlib.figure.AxesImage at 0x1711a2ac3c8>
```



Execute o código da celula abaixo para classificar com o modelo AlexNet a imagem tratada

```
[92]: image = img.to(device).unsqueeze(0)
      output = modelAlexNet(image)
      _, pred = torch.max(output, 1)
      print(classes[pred.item()])
```

```
formiga
```

Batch de 20 imagens de validação submetido ao modelo AlexNet treinado anteriormente com NLLL e Otimizador ADAM

Na celula abaixo é submetido um Batch de 20 imagens ao modelo AlexNet treinado aneriormente

In [107]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelAlexNet(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

In [ ]:

Podemos conferir na imagem que o modelo acertou aproximadamente 47% das vezes

Testes do modelo AlexNet com Função de custo NLLL e Otimizador SGD

Treinamento em cinco épocas e validação do modelo AlexNet com Função de custo NLLL e Otimizador SGD

Para testar o modelo AlexNet com Função de custo NLLL e Otimizador SGD

In [108]:
```
# Para o modelo AlexNet
# Função de custo NLLL e Otimizador SGD
criterion = criterionNLLL
optimizer = optimizerSGDAlexNet
epochs = 5

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: -0.0078, acc 0.5123
validation loss: -0.0235, validation acc 0.5621
epoch : 2
training loss: -0.0408, acc 0.5164
validation loss: -0.0548, validation acc 0.5948
epoch : 3
training loss: -0.0788, acc 0.5246
validation loss: -0.0901, validation acc 0.6405
epoch : 4
training loss: -0.1108, acc 0.5697
validation loss: -0.1220, validation acc 0.6601
epoch : 5
training loss: -0.1539, acc 0.6066
validation loss: -0.1581, validation acc 0.6797
```

Para testar o modelo AlexNet com Função de custo NLLL e Otimizador SGD

In [109]:
```
# Para o modelo AlexNet
# Função de custo NLLL e Otimizador SGD
criterion = criterionNLLL
optimizer = optimizerSGDAlexNet
epochs = 10

# Executa o treinamento do modelo
Treinamento(optimizer, criterion, modelAlexNet, epochs)
```

```
epoch : 1
training loss: -0.1916, acc 0.5861
validation loss: -0.1987, validation acc 0.7190
epoch : 2
training loss: -0.2372, acc 0.6557
validation loss: -0.2351, validation acc 0.6928
epoch : 3
training loss: -0.2754, acc 0.5984
validation loss: -0.2779, validation acc 0.6797
epoch : 4
training loss: -0.3258, acc 0.6721
validation loss: -0.3194, validation acc 0.7255
epoch : 5
training loss: -0.3530, acc 0.6107
validation loss: -0.3634, validation acc 0.6993
epoch : 6
training loss: -0.4210, acc 0.6148
validation loss: -0.4020, validation acc 0.7124
epoch : 7
training loss: -0.4592, acc 0.6721
validation loss: -0.4515, validation acc 0.6732
epoch : 8
training loss: -0.5144, acc 0.6475
validation loss: -0.4983, validation acc 0.6993
epoch : 9
training loss: -0.5642, acc 0.6762
validation loss: -0.5675, validation acc 0.7320
epoch : 10
training loss: -0.6411, acc 0.6762
validation loss: -0.6244, validation acc 0.7124
```

Como o resultado em 5 épocas processou rapidamente, decidi aumentar para 10 épocas, com isso o resultado melhorou levemente.

Gráfico comparativo dos valores de custo do treinamento e validação do modelo AlexNet com Função de custo NLLL e Otimizador SGD

Grafico comparativo dos valores de custo do treinamento e validação

In [110]:
```
plt.plot(running_loss_history, label='training loss')
plt.plot(val_running_loss_history, label='validation loss')
plt.legend()
```

Out[110]: <matplotlib.legend.Legend at 0x1711695e320>

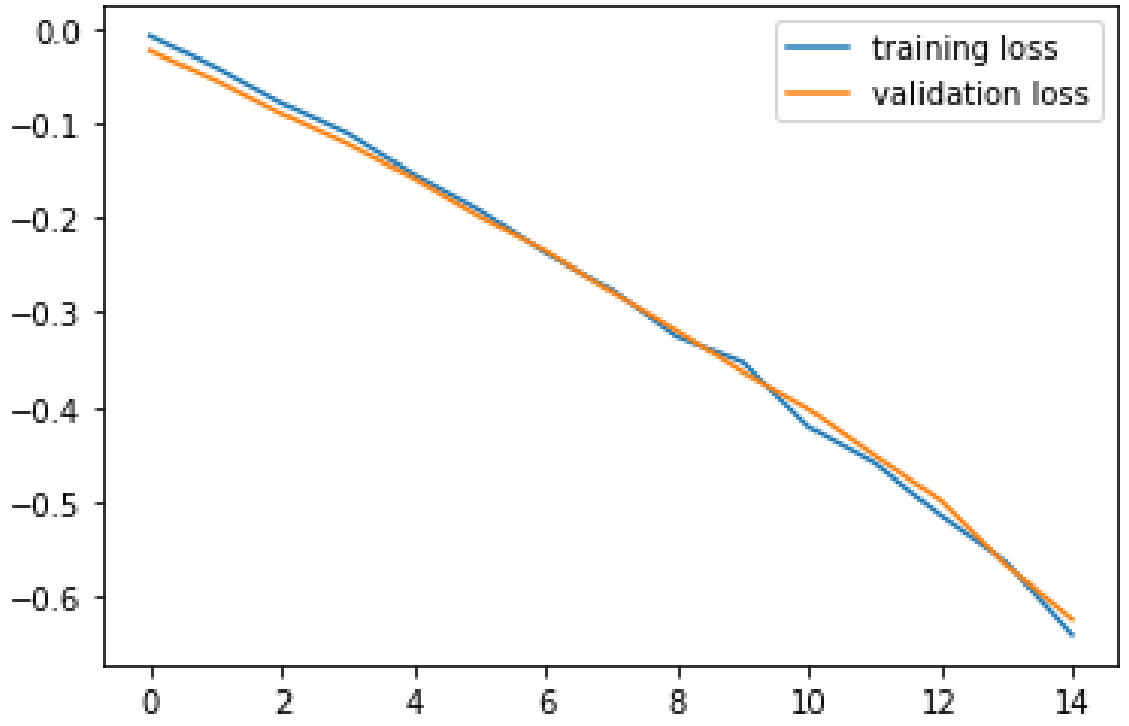
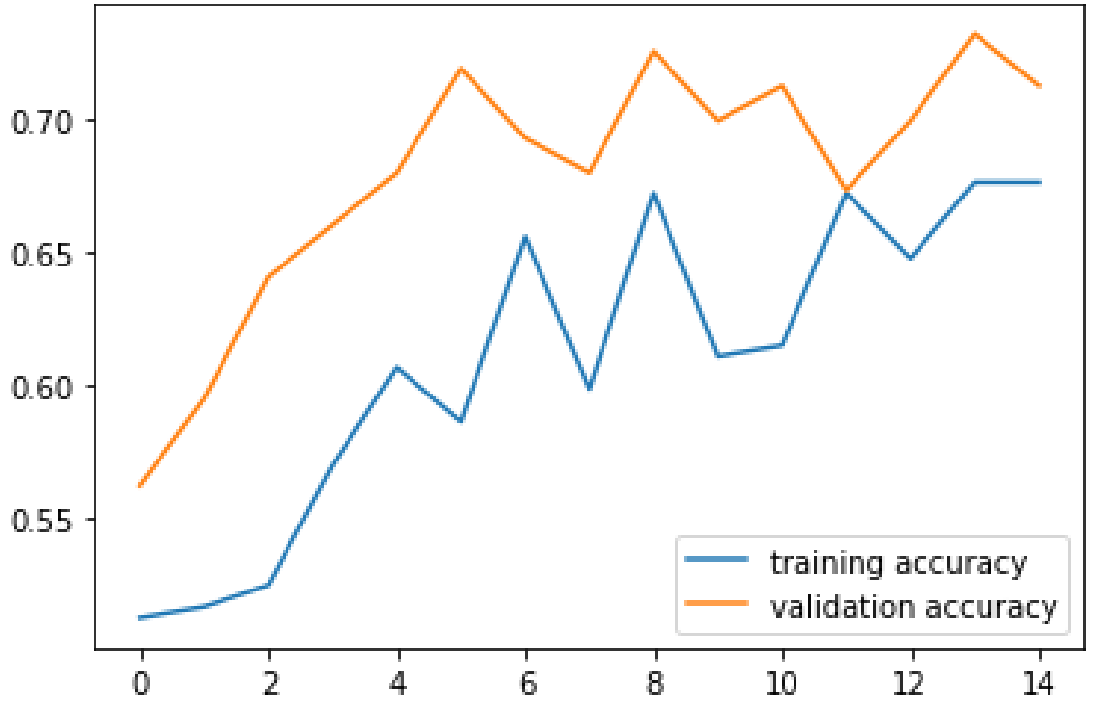


Gráfico comparativo dos valores de acurácia do treinamento e validação do modelo AlexNet com Função de custo NLLL e Otimizador SGD

Grafico comparativo dos valores de acuracia do treinamento e validação

In [112]:
```
plt.plot(running_corrects_history, label='training accuracy')
plt.plot(val_running_corrects_history, label='validation accuracy')
plt.legend()
```

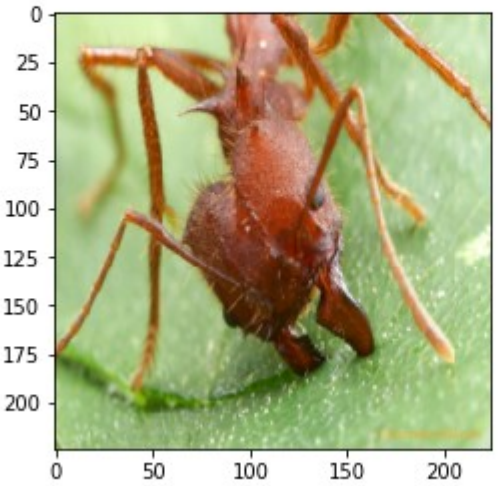
Out[112]:
```
<matplotlib.legend.Legend at 0x17116a50a20>
```



Resultado da validação do modelo AlexNet com Função de custo NLLL e Otimizador SGD com uma imagem da internet

In [114]:
```
img = transform_validation(img)
plt.imshow(im_convert(img))
```

Out[114]:
```
<matplotlib.image.AxesImage at 0x17119cba208>
```



Execute o código da celula abaixo para classificar com o modelo AlexNet a imagem

In [115]:
```
image = img.to(device).unsqueeze(0)
output = modelAlexNet(image)
_, pred = torch.max(output, 1)
print(classes[pred.item()])
```

```
formiga
```

Batch de 20 imagens de validação submetido ao modelo AlexNet treinado anteriormente com NLLL e Otimizador SGD

Na celula abaixo é submetido um Batch de 20 imagens ao modelo AlexNet treinado aneriormente

n [116]:
```
dataiter = iter(validation_loader)
images, labels = dataiter.next()
images = images.to(device)
labels = labels.to(device)
output = modelAlexNet(images)
_, preds = torch.max(output, 1)

fig = plt.figure(figsize=(25, 4))

for idx in np.arange(20):
  ax = fig.add_subplot(2, 10, idx+1, xticks=[], yticks=[])
  plt.imshow(im_convert(images[idx]))
  ax.set_title("{} ({})".format(str(classes[preds[idx].item()]), str(classes[labels[idx].item()])), color=("green" if preds[i
```

In [ ]:

Podemos conferir na imagem que o modelo acertou aproximadamente 60% das vezes

Conclusões

Após os testes foi possível concluir que a escolha dos hyperparametros fez diferença significativa no resultado obtido, observável através dos resultados do teste do Modelo VGG16 com função de Custo CrossEntropy e Otimizador SGD aproximadamente 94% de acurácia contra o modelo AlexNet com Função de custo NLLL e Otimizador ADAM com aproximadamente 48% de acurácia