**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL  
CENTRO DE EDUCAÇÃO CONTINUADA**

**ESPECIALIZAÇÃO EM CIÊNCIA DE DADOS**

**LEANDRO ROSSETTI DE SOUZA**

**TRABALHO DP1**

**Porto Alegre**

**2019**

**LEANDRO ROSSETTI DE SOUZA**

**Trabalho DP1:**

* Compreender o protocolo completo de treinamento, validação e avaliação de redes profundas e/ou recorrentes.
* Compreender o impacto de diferentes escolhas de arquiteturas e hiperparâmetros no processo de treinamento.
* Compreender as possibilidades existentes no uso das informações obtidas através de ConvNets.
* Ter experiência com um framework de Deep Learning.
* Executar experimentos de acordo com uma metodologia.
* Avaliar os resultados obtidos

Trabalho apresentado como requisito para

compor a avaliação na disciplina Deep Learning I

Professor: Prof.º. Jônatas Wehrmann

Link do GuitHub: <https://github.com/leandroro7/DP1_Trabalho>

Porto Alegre 2019

1. Definição do problema.

Para atender ao objetivo do trabalho, inicialmente foi definido o problema a ser tradado, e a partir dessa definição foram definidas as etapas necessárias para atingir o objetivo proposto.

O problema proposto foi treinar uma rede convolucional para o reconhecimento de imagens de Abelhas e Formigas, e a partir dessa definição iniciou-se o processo de pesquisa e desenvolvimento do modelo apresentado nesse trabalho.

1.1 Modelo de rede convolucional utilizado

Na pesquisa inicial foram definidos dois modelos de redes convolucionais para utilização nesse trabalho, o critério de escolha utilizado foi o seguinte:

1. O modelo deveria se enquadrar no conceito de rede convolucional, sendo capaz de executar a classificação de imagens;
2. O modelo deveria ser econômico em termos de consumo de recursos para processamento
3. O modelo deveria ser fácil de implementar no Pytorch

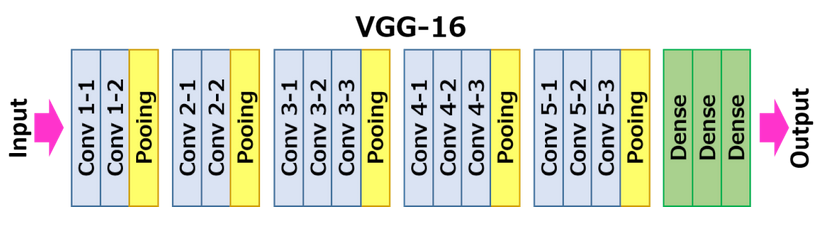
A partir dos critérios de seleção de modelos definidos, iniciei a pesquisa sobre quais modelos existentes se enquadrariam nesses critérios. Na comparação entre vários modelos existentes selecionei os dois, que ao meu ver, atendiam os requisitos, foram eles:

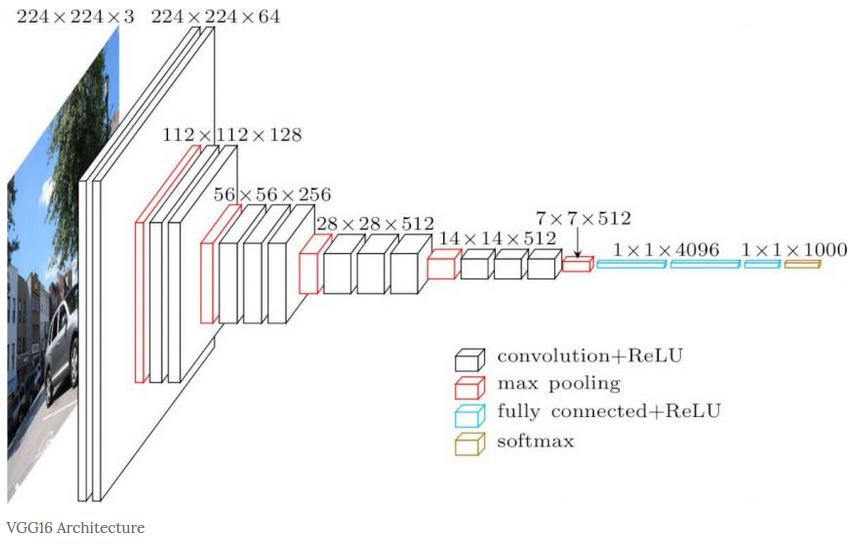
1. Modelo VGG16
2. Modelo AlexNet

1.2 Modelo VGG16

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 tamanho de um núcleo (11 e 5 na primeira e segunda camada convolucional, respectivamente) por vários filtros 3 × 3 do tamanho de um núcleo, um após o outro. O VGG16 foi treinado por semanas e estava usando GPUs NVIDIA Titan Black.

A arquitetura mostrada abaixo é uma representação do modelo VGG16.





A entrada para a camada cov1 é uma imagem fixa de tamanho 224 x 224 RGB. A imagem é passada através de uma pilha de camadas convolucionais (conv.), Onde os filtros foram usados ​​com um campo receptivo muito pequeno: 3 × 3 (que é o menor tamanho para capturar a noção de esquerda / direita, cima / baixo, centro) ) Em uma das configurações, ele também utiliza filtros de convolução 1 × 1, que podem ser vistos como uma transformação linear dos canais de entrada (seguidos de não linearidade). O passo da convolução é fixo em 1 pixel; o preenchimento espacial de conv. a entrada da camada é tal que a resolução espacial é preservada após a convolução, ou seja, o preenchimento é de 1 pixel para 3 × 3 conv. camadas. O agrupamento espacial é realizado por cinco camadas de agrupamento máximo, que seguem parte da conv. camadas (nem todas as camadas conv. são seguidas pelo pool máximo). O pool máximo é realizado em uma janela de 2 × 2 pixels, com passo 2.

Três camadas totalmente conectadas (FC) seguem uma pilha de camadas convolucionais (que tem uma profundidade diferente em arquiteturas diferentes): as duas primeiras têm 4096 canais cada, a terceira executa a classificação ILSVRC de 1000 vias e, portanto, contém 1000 canais (um para cada classe). A camada final é a camada soft-max. A configuração das camadas totalmente conectadas é a mesma em todas as redes. Todas as camadas ocultas são equipadas com a não linearidade da retificação (ReLU). Observe também que nenhuma das redes (exceto uma) contém Normalização de Resposta Local (LRN), essa normalização não melhora o desempenho no conjunto de dados ILSVRC, mas leva ao aumento do consumo de memória e do tempo de computação.

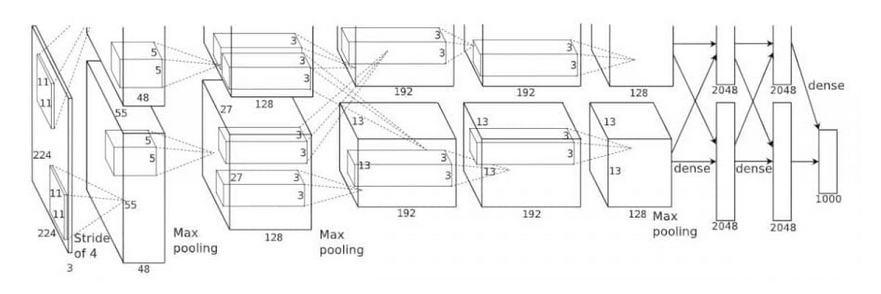
1.3 Modelo AlexNet

AlexNet é o nome de uma rede neural convolucional que teve um grande impacto no campo de aprendizado de máquina, especificamente na aplicação de aprendizado profundo à visão de máquina. Ele venceu a competição 2012 ImageNet LSVRC-2012 por uma grande margem (taxas de erro de 15,3% VS 26,2% (segundo lugar)). A rede tinha uma arquitetura muito semelhante à LeNet de Yann LeCun et al, mas era mais profunda, com mais filtros por camada e com camadas convolucionais empilhadas. Consistia em 11 × 11, 5 × 5,3 × 3, convoluções, pool máximo, desistência, aumento de dados, ativações ReLU, SGD com momento. Ele anexou as ativações da ReLU após cada camada convolucional e totalmente conectada. A AlexNet foi treinada por 6 dias simultaneamente em duas GPUs Nvidia Geforce GTX 580, razão pela qual sua rede está dividida em dois pipelines.

Principais aspectos do modelo AlexNet

* A função de ativação Relu é usada em vez de Tanh para adicionar não linearidade. Acelera a velocidade em 6 vezes com a mesma precisão.
* Use a desistência em vez da regularização para lidar com o ajuste excessivo. No entanto, o tempo de treinamento é dobrado com a taxa de abandono de 0,5.
* Sobreposição de pool para reduzir o tamanho da rede. Reduz as taxas de erro top 1 e top 5 em 0,4% e 0,3%, respectivamente.

A arquitetura mostrada abaixo é uma representação do modelo AlexNet.



A arquitetura representada acima, o modelo AlexNet, contém oito camadas com pesos; os cinco primeiros são convolucionais e os três restantes estão totalmente conectados. A saída da última camada totalmente conectada é alimentada com um softmax de 1000 vias, que produz uma distribuição pelas etiquetas da classe 1000. A rede maximiza o objetivo da regressão logística multinomial, que é equivalente a maximizar a média entre os casos de treinamento da probabilidade logarítmica do rótulo correto na distribuição de previsão. Os kernels da segunda, quarta e quinta camadas convolucionais são conectados apenas aos mapas de kernel na camada anterior que residem na mesma GPU. Os núcleos da terceira camada convolucional são conectados a todos os mapas do kernel na segunda camada. Os neurônios nas camadas totalmente conectadas são conectados a todos os neurônios na camada anterior.

1.4 Justificativa da escolha dos modelos

A facilidade de implementação e o baixo custo de processamento foram os pontos chave para minha escolha.

O baixo custo de processamento foi obtido ao optar pelo uso do método de Transfer Learning, pois em ambos os modelos é possível utilizar a opção *pretrained=True,* com isso eu poderia aproveitar o conhecimento acumulado de modelos pré treinados com o datasheet Imagenet, pulando a custosa etapa de processamento para extração de features, e com isso utilizar minha própria infraestrutura, um notebook Deel Inspiron com processador I7 e 16Gb de memória ram, para especializar esses modelos para reconhecimento das imagens de Abelhas e Formigas.

1. Metodologia

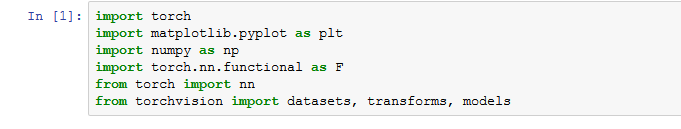
O trabalho em questão pode ser classificado como uma pesquisa exploratória. A pesquisa exploratória, segundo Cervo e Silva (2006), estabelece critérios, métodos e técnicas para a elaboração de uma pesquisa e visa oferecer informações sobre o objeto desta e orientar a formulação de hipóteses.

1. Desenvolvimento do projeto

Para execução do projeto, iniciei montando o dataset de imagens, que foram obtidas na internet e separadas em pastas especificas para Abelhas e Formigas; a partir disso iniciei a definição dos casos de teste baseados nos modelos e as possíveis configurações de hiperparametros. Com os casos de teste definidos foram então definidas as validações que iria executar nos resultado obtidos.

3.1 Desenvolvimento do código

Iniciei o desenvolvimento em um jupyter notebook definindo os pacotes utilizados



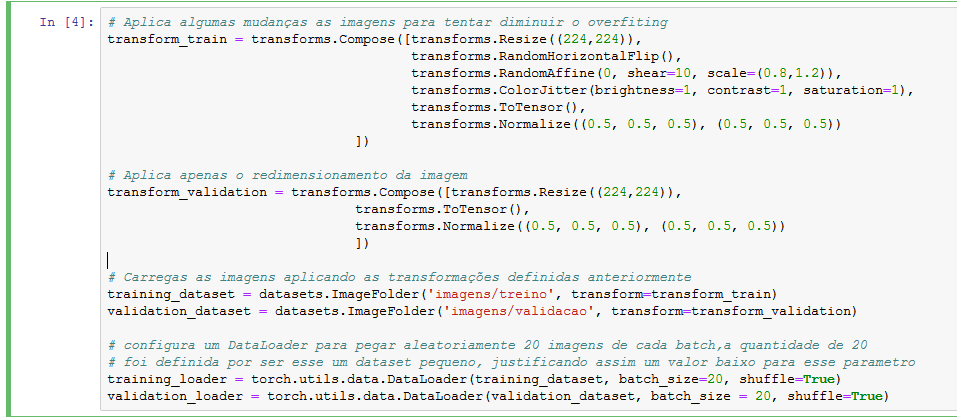
Em seguida verifico se existe uma GPU na máquina, e caso exista o código abaixo utilizara a mesma para o processamento

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

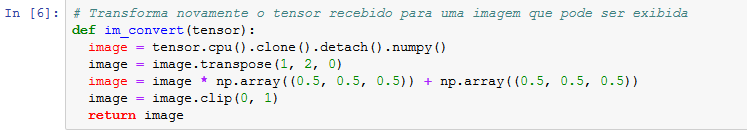
Criei o dataset de imagens e disponibilizei o mesmo no GitHub, abaixo está o código responsável por copia-lo para a maquina de execução dos modelos:

!git clone https://github.com/leandroro7/imagens.git

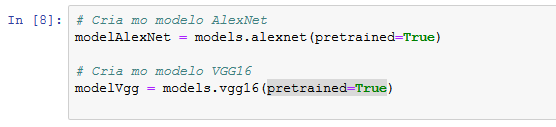
Após a cópia do dataset para a maquina local, aplico algumas transformações nas imagens e crio os dataloaders, funções para carregar os datasets em batchs de 20 imagens, que serão utilizados ao longo do código.



Os dataloaders criados carregam as imagens como tensores, e para exibi-las como imagens em um gráfico é necessário converte-as novamente, isso é feito com a função de conversão abaixo:



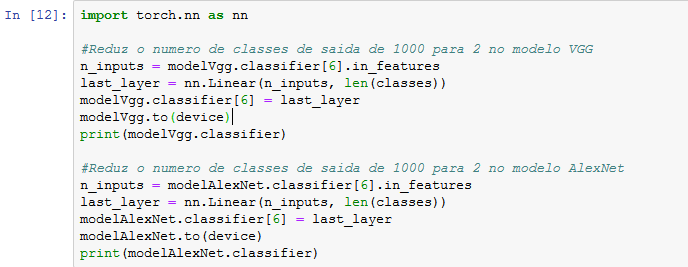
Após a definição do dataset passo a criação dos modelos, conforme dito anteriormente, os modelos utilizados foram configurados com a opção “preteined = True”



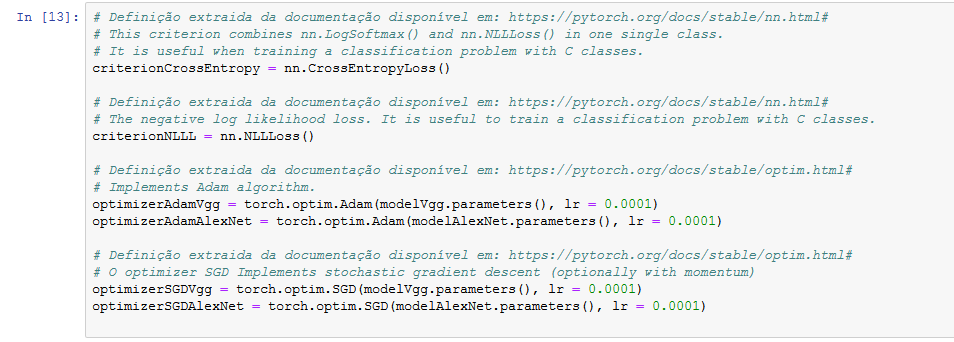
Como estou utilizando modelos pré-treinados, foi necessário preservar a parte convolucional do modelo, na célula abaixo utilizo um loop para "congelar" a parte de extração de features não perder o "conhecimento" já acumulado.



Originalmente esses modelos foram treinados com o dataset Imagenet, que contem mil classes na camada de saída, na imagem abaixo é feito o ajuste da camada de saída, pois agora existem apenas duas classes para serem classificadas



Para fins de comparação entre modelos foram definidos para teste nesse trabalho duas funções de custo [CrossEntropy, NLLLoss] e dois otimizadores [Adam, SGD], esse conjunto de hiperparametros possibilitou a geração de quatro configurações distintas para cada modelo, gerando um total de oito casos de teste validados e descritos na sequência desse trabalho. Na imagem abaixo está a definição desses hiperparametros:



O treinamento foi definido em uma função, que é utilizado ao longo do código nos casos de teste. A função foi definida conforme segue:



3.2 Definição do Conjunto de testes

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

* 1. 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
  2. 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 executadas oito rodadas de testes cobrindo todos os modelos e hyperparametros definidos acima.

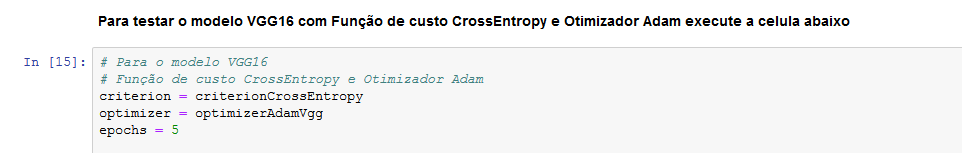
1. Execução do Treinamento

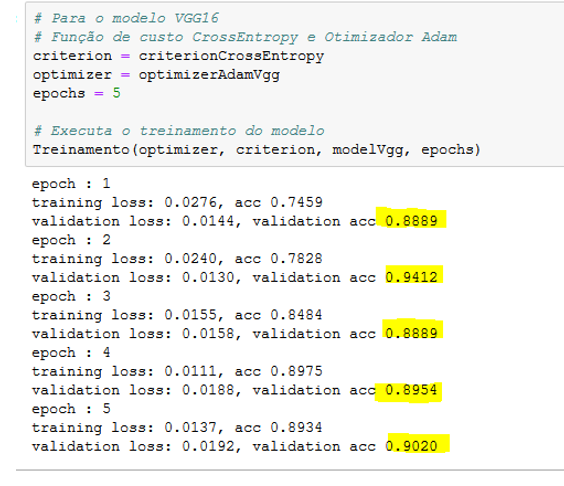
Os testes foram executados utilizando imagens da internet, e os resultados obtidos foram de acordo com os valores de acurácia preditos no treinamento.

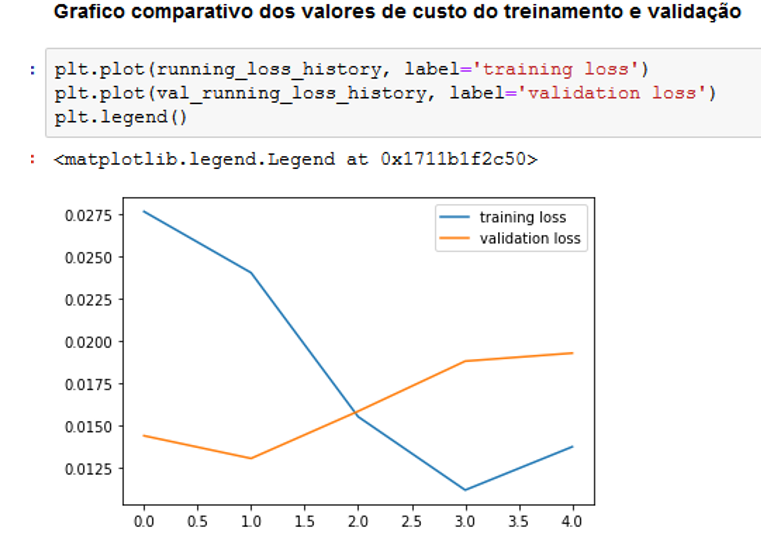
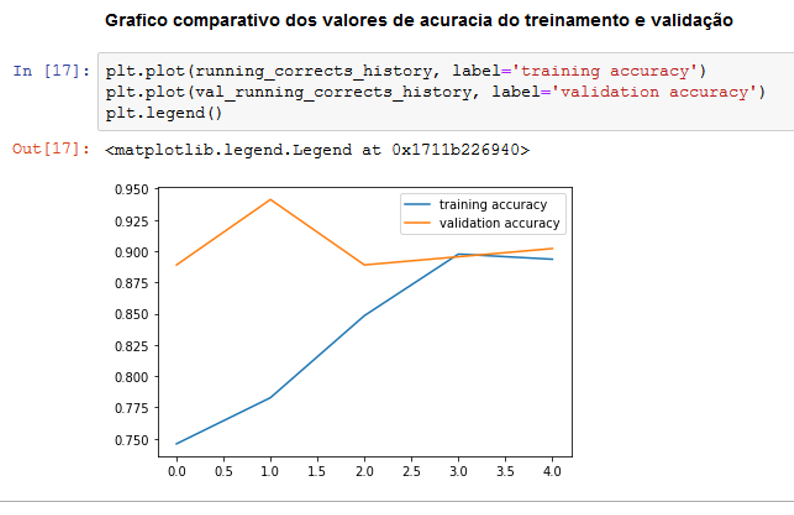
Abaixo descrevo a sequência dos testes realizados.

4.1 Treinamento do modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 90% na validação

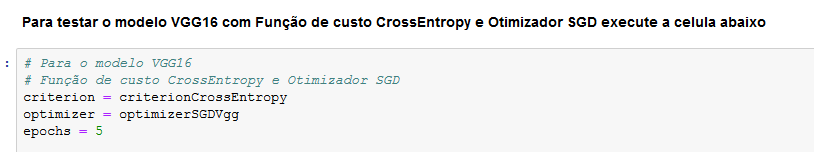


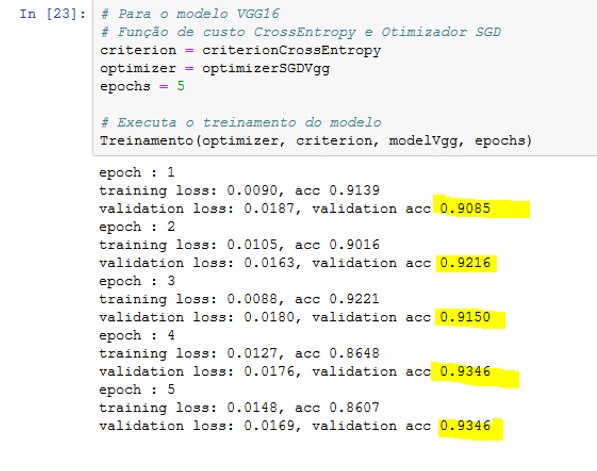


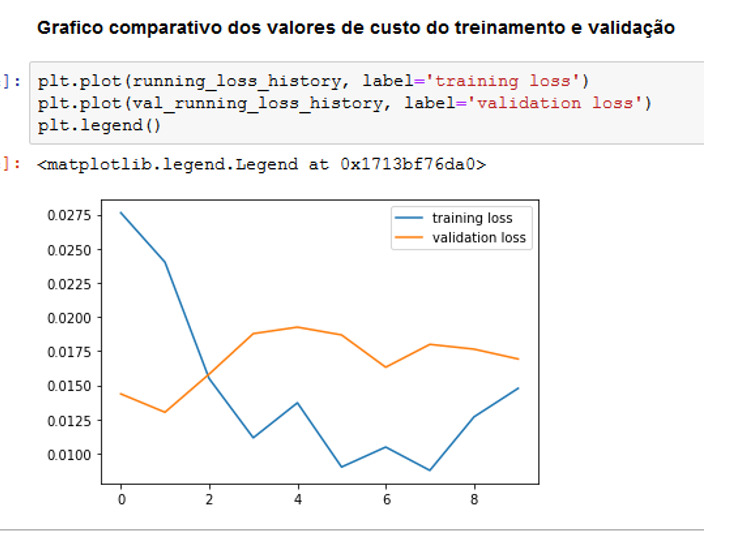
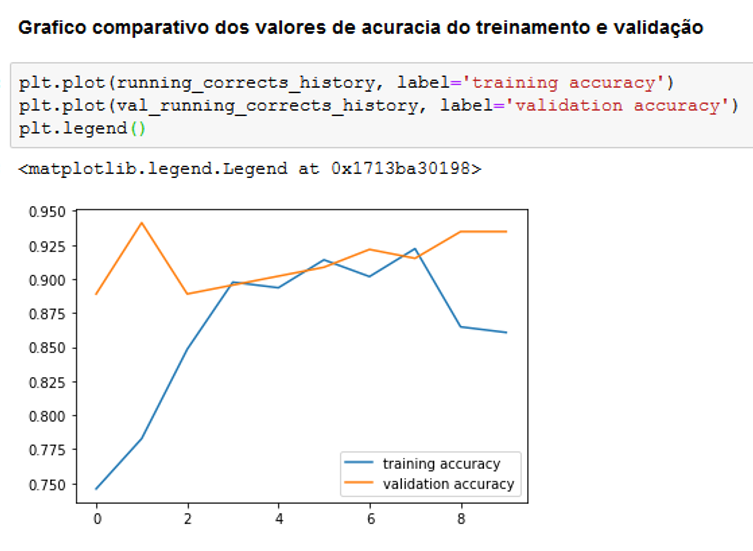
 

4.2 Treinamento do modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 93% na validação



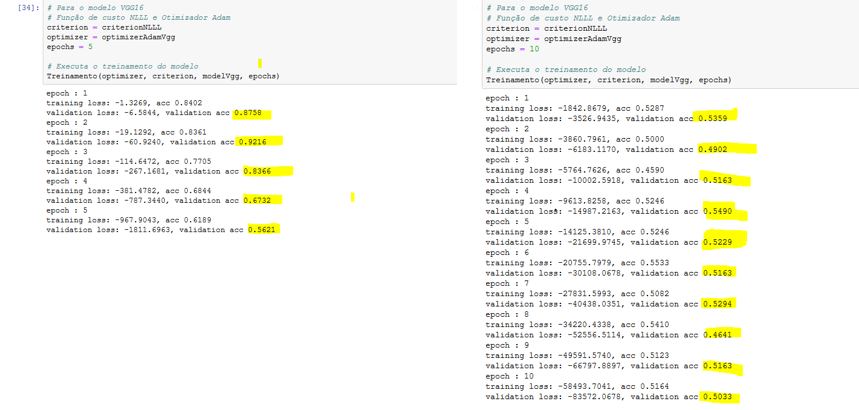


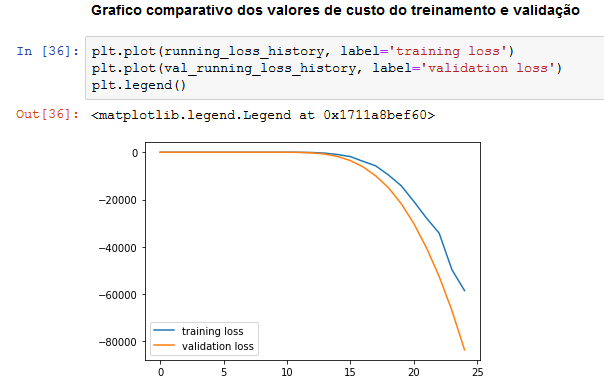
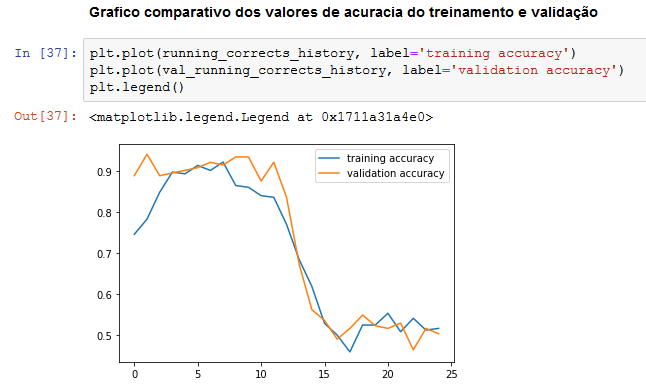
 

4.3 Treinamento do modelo VGG16 com Função de custo NLLL e Otimizador Adam

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 50% na validação





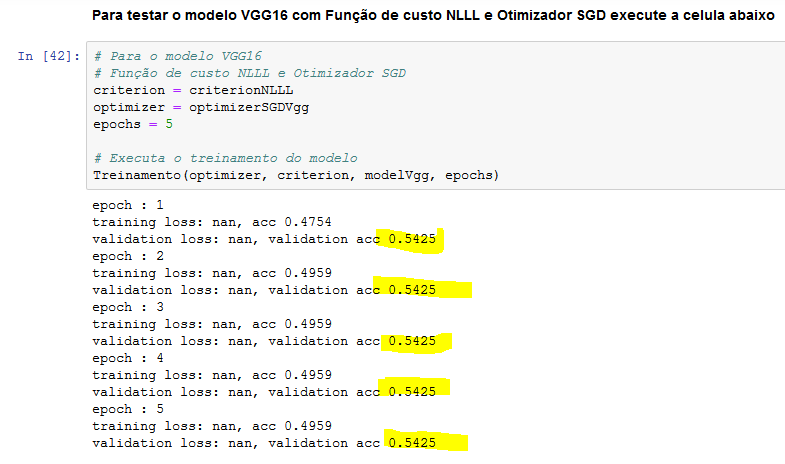
 

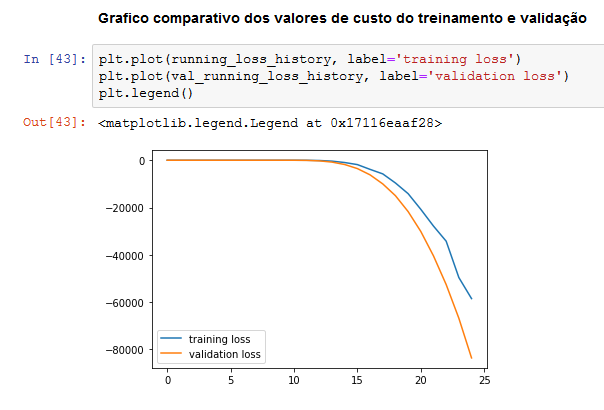
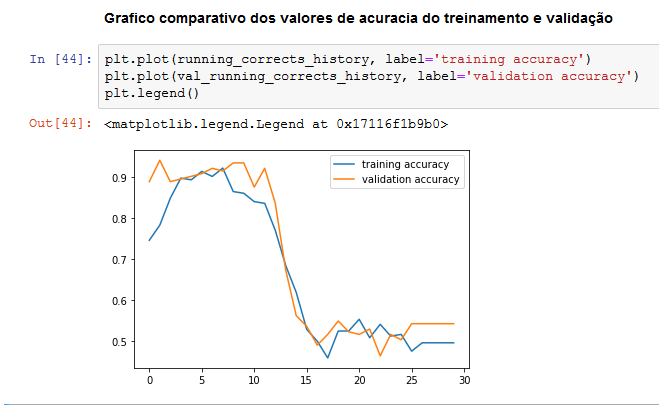
Como o resultado em 5 épocas foi muito fraco, decidi aumentar para 10 épocas, porem o resultado piorou.

4.4 Treinamento do modelo VGG16 com Função de custo NLLL e Otimizador SGD

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 54% na validação

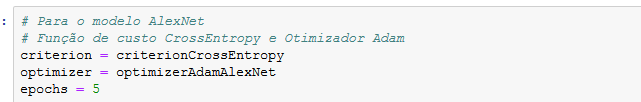


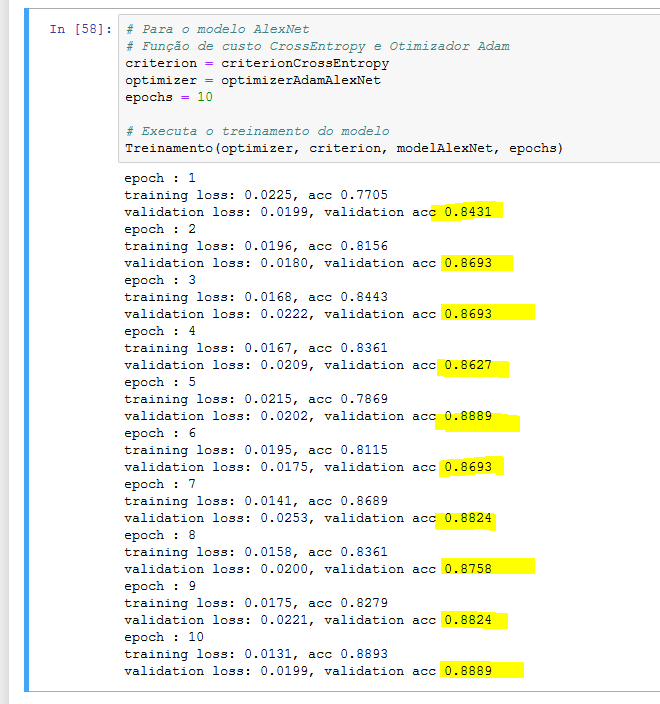


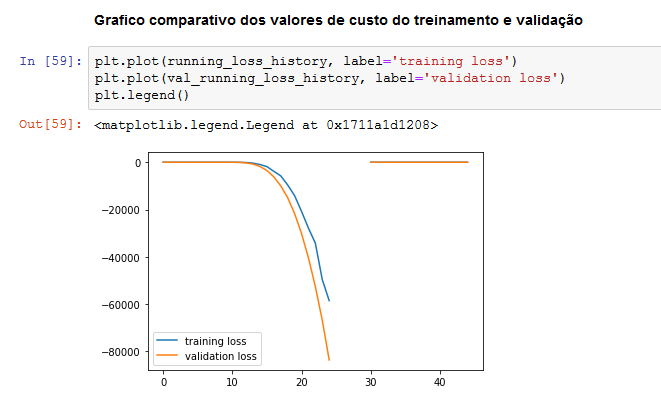
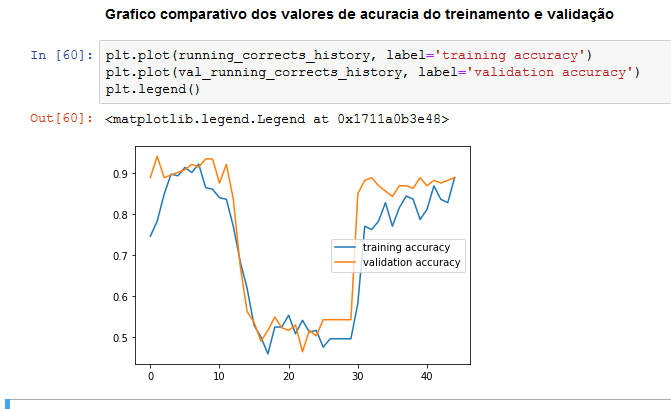
 

4.5 Treinamento do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 88% na validação



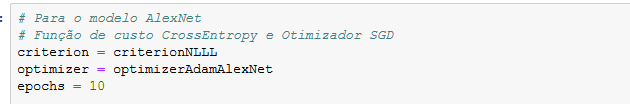


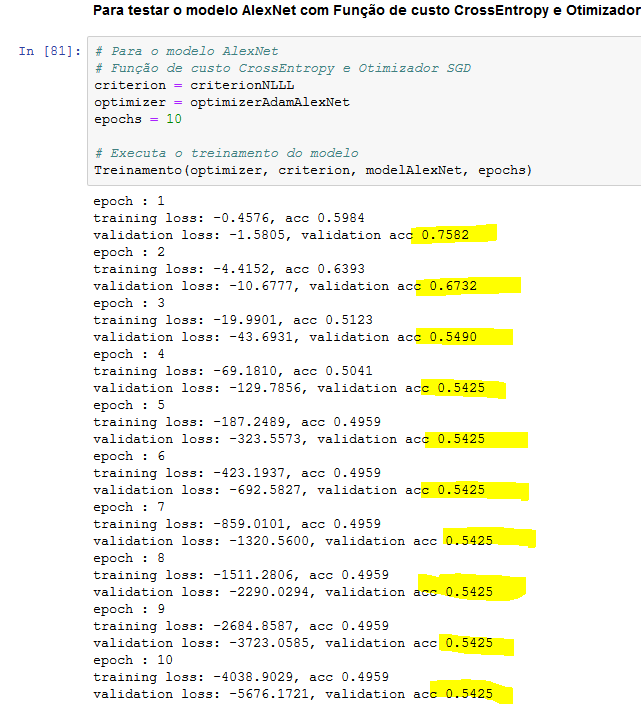
 

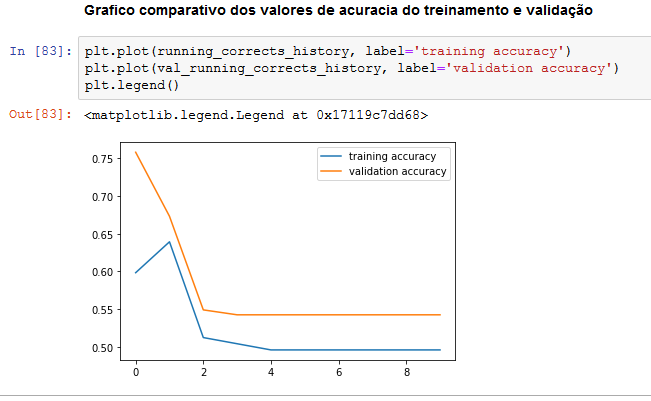
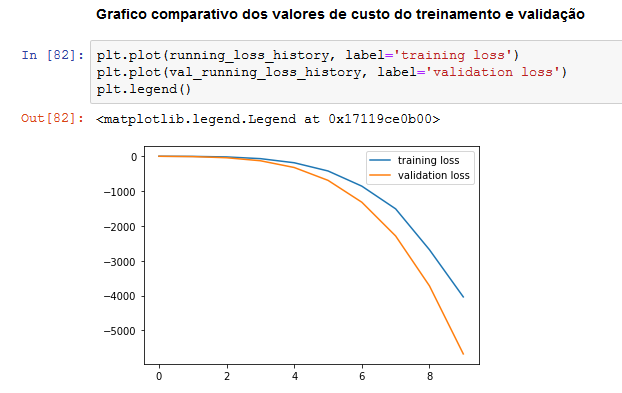
Como o resultado em 5 épocas processou rapidamente, decidi aumentar para 10 épocas, com isso o resultado melhorou aproximadamente 20%

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

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 54% na validação



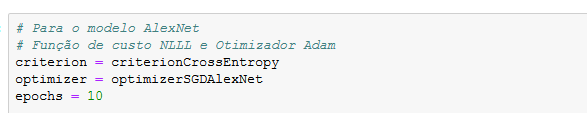


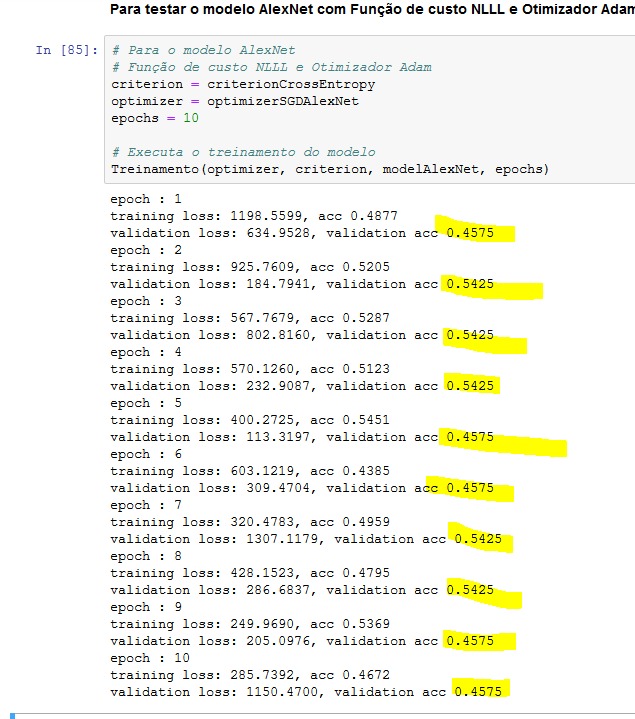


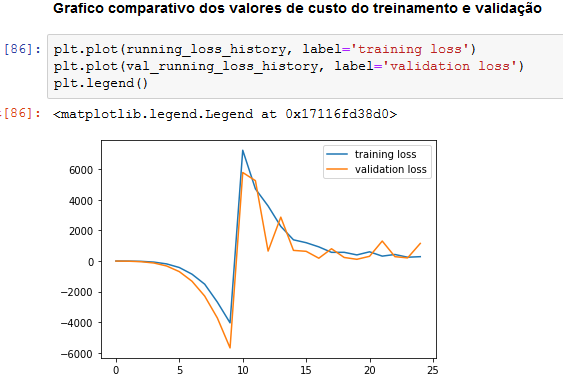
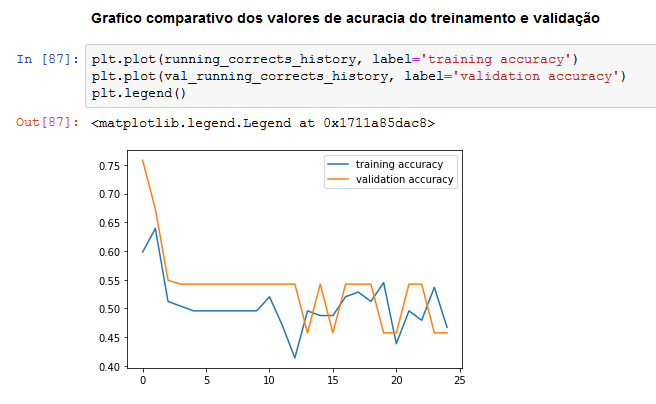
Como o resultado em 5 épocas processou rapidamente, decidi aumentar para 10 épocas, porem diferente com o otimizador ADAM, com o otimizados SGD resultado não melhorou significativamente.

4.7 Treinamento do modelo AlexNet com Função de custo NLLL e Otimizador Adam

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 46% na validação



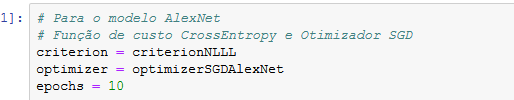


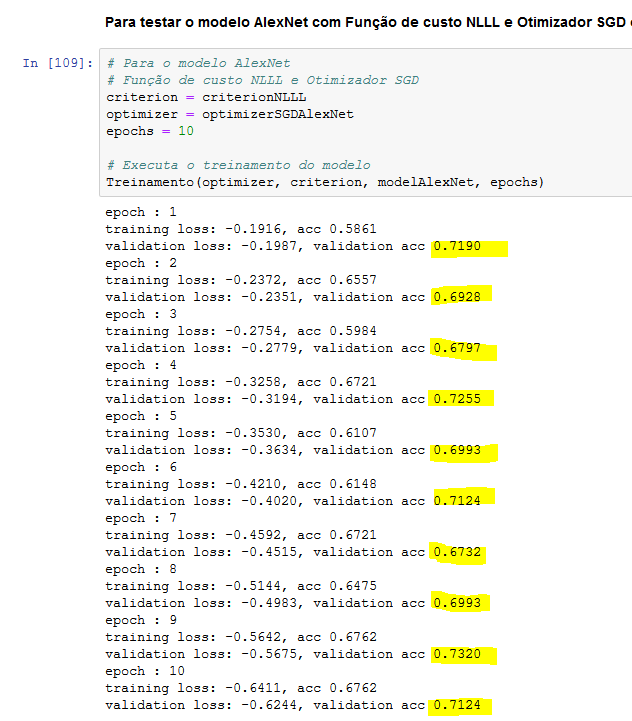
 

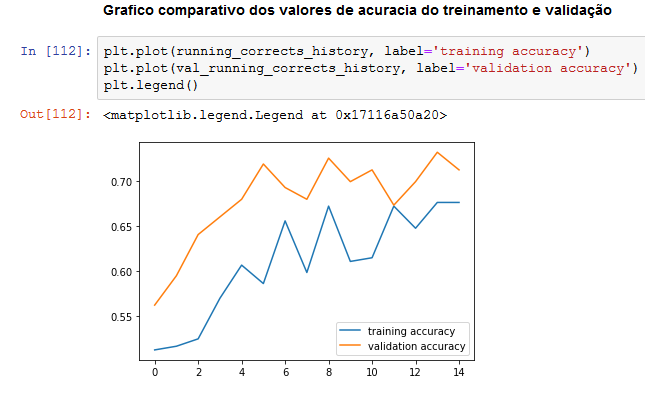
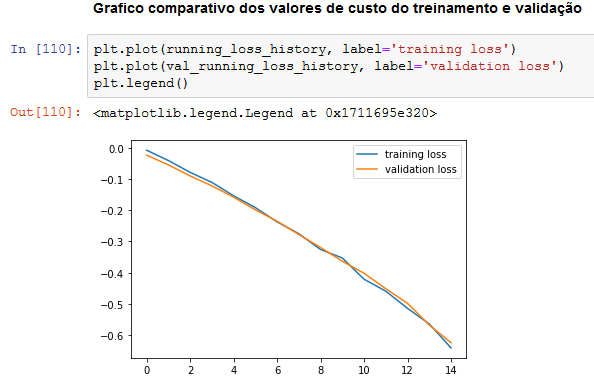
Como o resultado em 5 épocas processou rapidamente, decidi aumentar para 10 épocas, com o otimizador ADAM o resultado não melhorou, ao contrário ele diminuiu 10% de acurácia passando de 54% para 45%.

4.8 Treinamento do modelo AlexNet com Função de custo NLLL e Otimizador SGD

O treinamento foi executado em cinco épocas, e obteve-se uma acurácia de aproximadamente 71% na validação







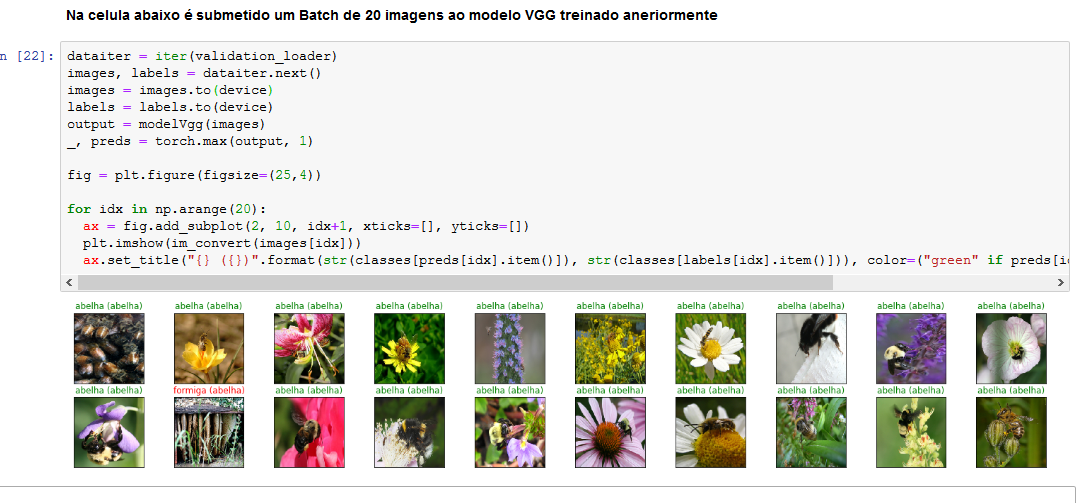
Como o resultado em 5 épocas processou rapidamente, decidi aumentar para 10 épocas, com isso o resultado melhorou levemente, aproximadamente 4%, passando de 67% para 71%.

1. Execução dos Testes

Foram executados testes com todos os modelos treinados anteriormente, abaixo segue o processo deteste e os resultados obtidos.

5.1 Teste do modelo VGG16 com Função de custo CrossEntropy e Otimizador Adam

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

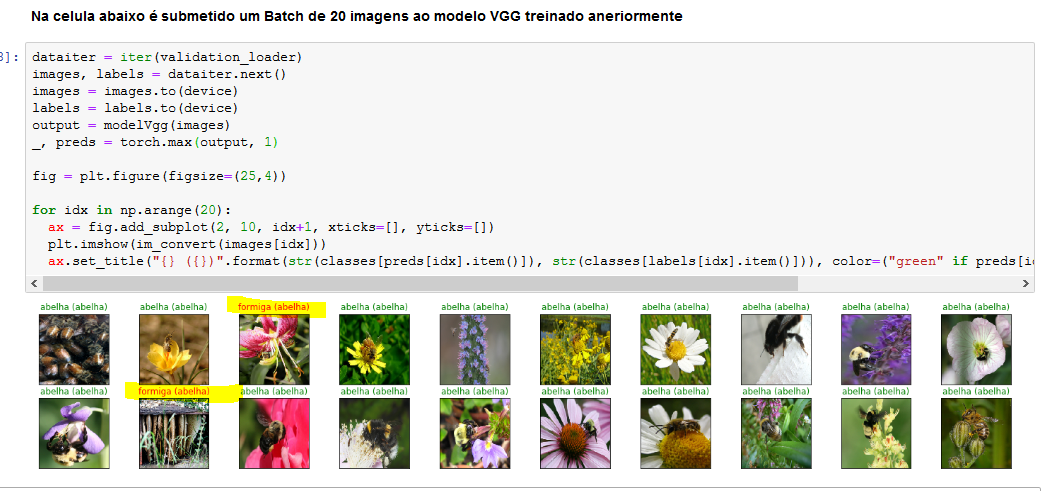


Resultado da validação do modelo VGG com Função de custo CrossEntropy e Otimizador Adam com uma imagem da internet



5.2 Teste do modelo VGG16 com Função de custo CrossEntropy e Otimizador SGD

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

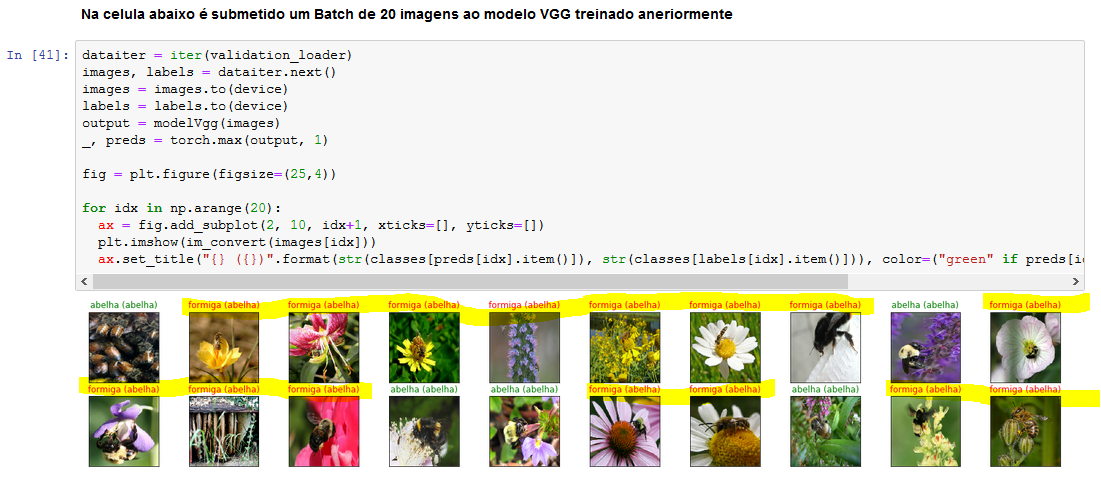


Resultado da validação do modelo VGG com Função de custo CrossEntropy e Otimizador SGD com uma imagem da internet



5.3 Teste do modelo VGG16 com Função de custo NLLL e Otimizador Adam

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

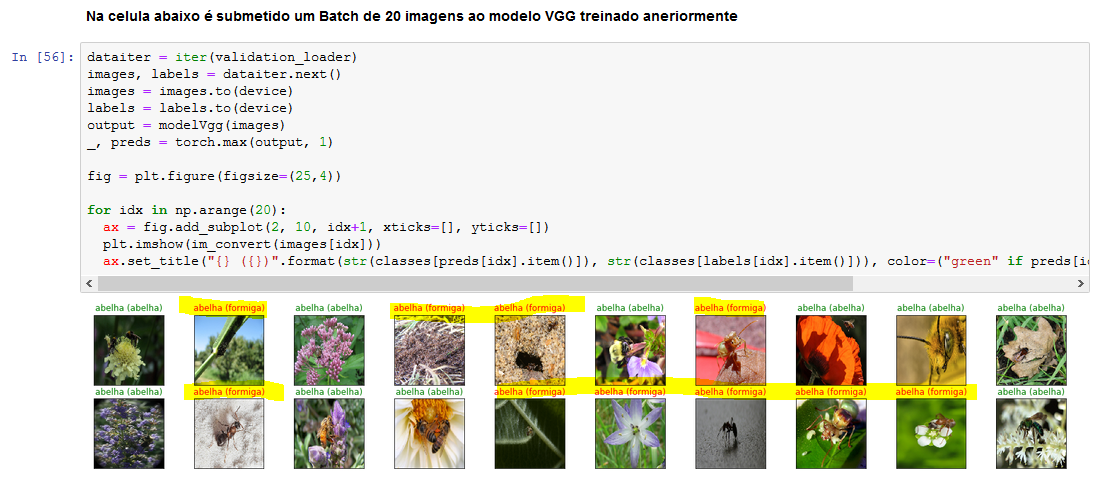


Resultado da validação do modelo VGG com Função de custo NLLL e Otimizador ADAM com uma imagem da internet

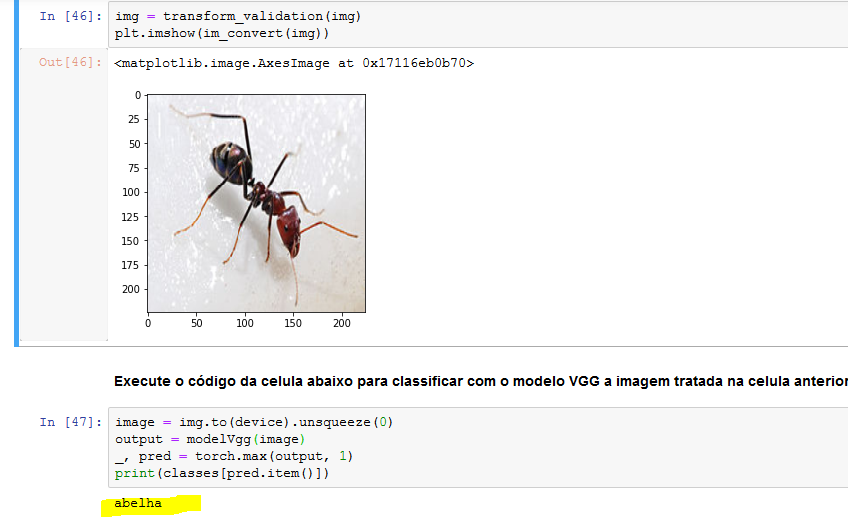


5.4 Teste do modelo VGG16 com Função de custo NLLL e Otimizador SGD

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

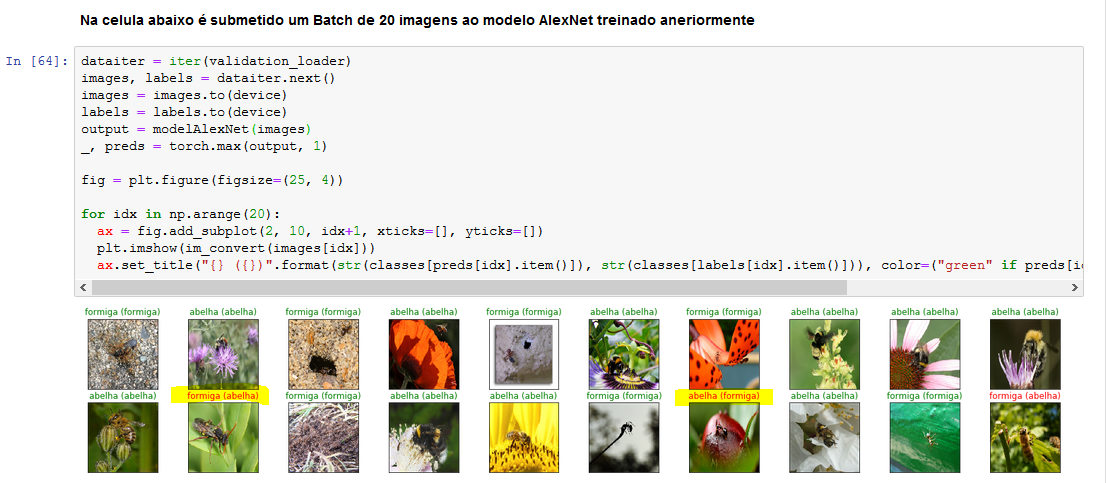


Resultado da validação do modelo VGG com Função de custo NLLL e Otimizador SGD com uma imagem da internet



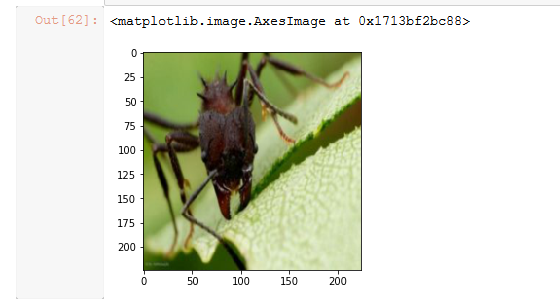
5.5 Teste do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam

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



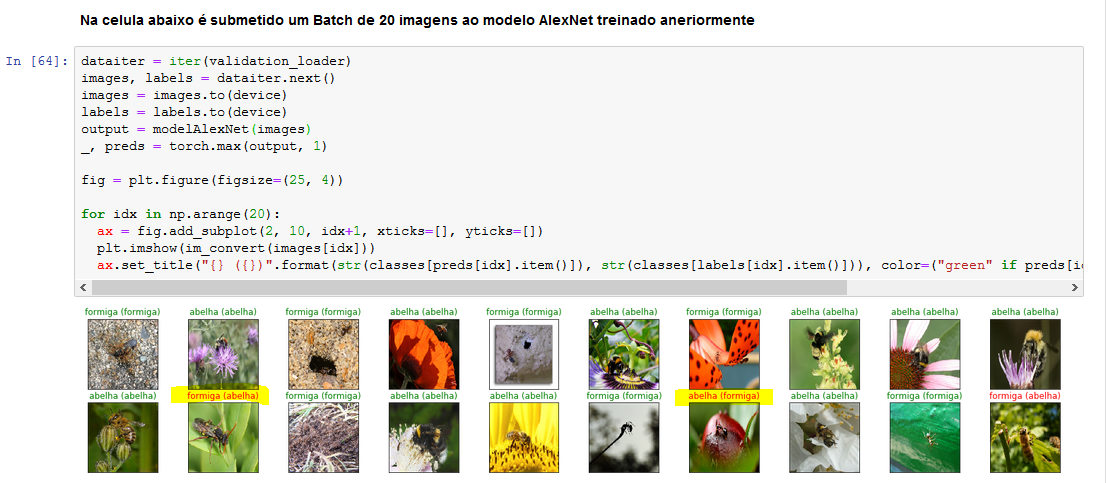
Resultado da validação do modelo AlexNet com Função de custo CrossEntropy e Otimizador Adam com uma imagem da internet





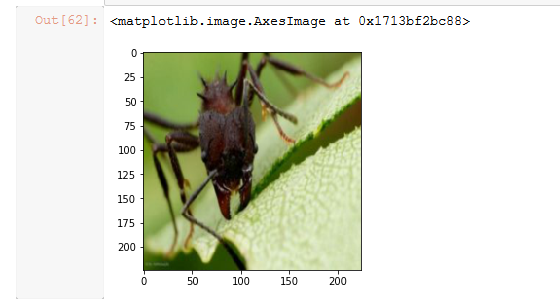
5.6 Teste do modelo AlexNet com Função de custo CrossEntropy e Otimizador SGD

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



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

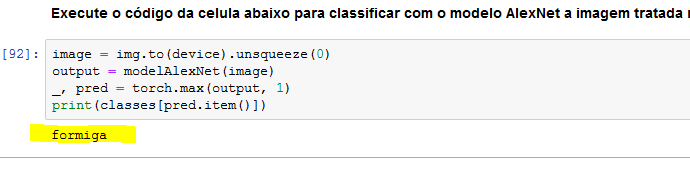


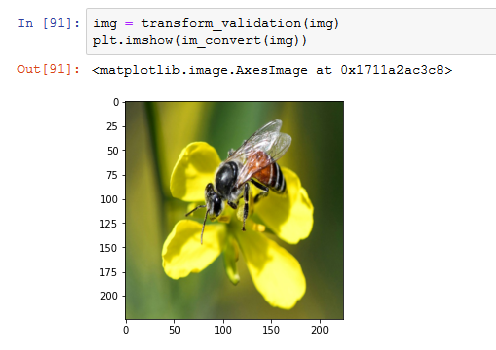


5.7 Teste do modelo AlexNet com Função de custo NLLL e Otimizador Adam

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



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



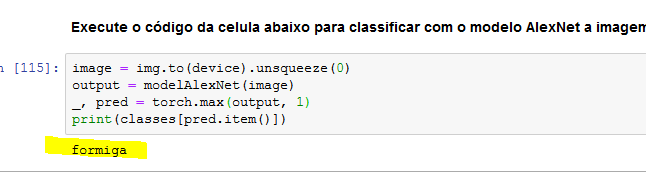
5.8 Teste do modelo AlexNet com Função de custo NLLL e Otimizador SGD

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



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





1. Conclusões

Ao longo do desenvolvimento foi possível colocar em prática alguns conhecimentos adquiridos na disciplina, um dos mais uteis e evidenciado nesse trabalho foi a necessidade de ajuste dos hierparametros.

Outro fato que ficou claro foi a diferença de resultados em uma rede com os mesmos hiperparametros porém com arquitetura (modelo) diferente

1. Referencias

Modelo VGG: <https://neurohive.io/en/popular-networks/vgg16/>

Modelo AlexNet: <https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/>