Relatório Deep Learning
Assignment 1: Transfer Learning

Ana Carolina Erthal & Felipe Lamarca

Link para o Google Colab

1 Modificações no código

Para o primeiro exercício, alteramos o parâmetro que inicializava o modelo com os pesos pré-treinados na ImageNet e passamos a treinar a rede apenas no novo conjunto de dados. A natureza dos dados também tornou necessário realizar alguns ajustes em relação à função de perda e de ativação: trata-se de um problema de classificação de 3 classes, e não de 2. Na prática, passamos a utilizar a `CategoricalCrossentropy()` como parâmetro da `loss` e experimentamos as funções de ativação para Softmax e ReLu, conforme discutido em outras ocasiões do relatório.

Como tínhamos um problema multiclasse e estávamos associando cada neurônio à predição de uma classe, utilizamos o *one hot encoding* ao importar os dados, através do parâmetro `label_mode = 'categorical'`. Por esse motivo, também ajustamos a camada de predição (`prediction_layer`) para que ela fosse uma camada densa com três neurônios, na qual é aplicada uma função de ativação.

Também foi necessário ajustar o tamanho das imagens, que estavam no formato `(160, 160, 3)` no exemplo, mas são do formato `(224, 224, 3)` no dataset utilizado.

No segundo exercício, para fazer o *feature extraction*, passamos a ImageNet como parâmetro de obter os pesos pré-treinados, e aplicamos um *freeze* em todas as camadas da VGG16. Os experimentos com as funções de ativação no primeiro exercício sugeriram uma melhor performance da Softmax, e por isso a utilizamos como função de ativação do classificador.

Por fim, para o terceiro exercício, mantivemos o pré-treinamento na ImageNet, mas alteramos a quantidade de camadas treináveis da VGG16 testando *unfreeze* de blocos conforme as orientações do exercício.

No mais, criamos uma função auxiliar chamada `increase_model()`, que recebe como parâmetros o modelo e a função de ativação desejada, apenas para facilitar a leitura do código e o desenvolvimento do assignment. Na prática, executa-se essencialmente o mesmo pipeline do exemplo inicial, apenas ajustando os hiperparâmetros informados na seção abaixo.

2 Hiperparâmetros

Para todos os resultados apresentados na tabela de resultados abaixo, utilizamos os seguintes parâmetros:

Hiperparâmetro	**Valor**
`learning rate`	0.001
`# epochs`	50
`batch size`	32
`early stopping patience`	15
`loss`	Categorical Cross Entropy
`optimizer`	Adam

Tabela 1 – Hiperparâmetros

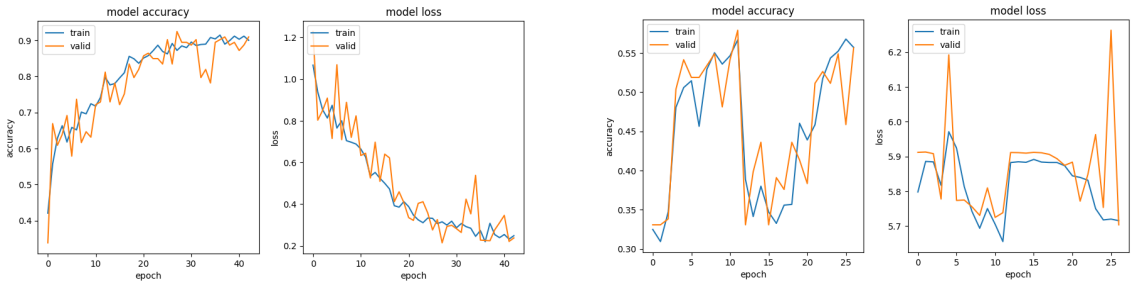
3 Resultados

Os resultados obtidos a partir dos modelos utilizados estão na tabela abaixo:

Exp.	**train acc**	**train loss**	**val acc**	**val loss**	**test acc**	**test loss**	**≈ train time**
1 (`Softmax`)	90.04%	0.2484	92.48%	0.2142	85.16%	0.3616	12′
1 (`ReLu`)	55.71%	5.7159	57.89%	5.7387	59.38%	5.5647	7′
2	79.50%	0.4854	83.46%	0.5555	73.44%	0.7321	5′
3-a	**99.81%**	**0.0120**	**98.50%**	**0.0568**	**95.31%**	0.1363	**3′**
3-b	98.94%	0.0328	97.74%	0.0718	94.53%	**0.1223**	5′
3-c	98.36%	0.0542	**98.50%**	0.0605	94.53%	0.1455	14′

Tabela 2 – Resultados

No primeiro exercício obtivemos um bom resultado, mostrando que o treinamento *from scratch* foi bastante eficaz. É interessante observar a diferença entre os resultados obtidos utilizando diferentes funções de ativação: fica evidente a prevalência da Softmax em relação à ReLu nesse contexto. Essa diferença se deve ao fato de que a ReLu não produz uma distribuição de probabilidade sobre as classes, importante para problemas multiclasse e, por esse motivo, ela é mais comumente utilizada em camadas ocultas da rede. Ainda assim, apresentamos esses resultados como experimentação.



(a) Softmax (b) ReLu

Figura 1 – Loss and Accuracy, Experiment 1

Quando realizamos o feature extractor no exercício 2, passando a fazer Transfer Learning a partir da ImageNet, é possível observar uma piora considerável em relação à nossa melhor tentativa no exercício 1 (da Softmax, que é o modelo que utilizamos para seguir em frente). Esse resultado é razoável e, de certa forma, esperado, porque nesse momento realizamos um *freeze* em todas as camadas da VGG16, de modo que a capacidade de aprendizagem do modelo em relação ao novo dataset é bastante limitada. Com as camadas da VGG16 sob *freeze*, os parâmetros foram estimados praticamente apenas através dos dados da ImageNet — provavelmente com menos imagens das categorias presentes no dataset utilizado. Por isso, é esperada uma acurácia menor e uma loss maior.

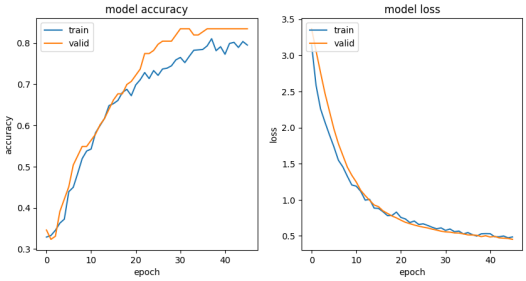
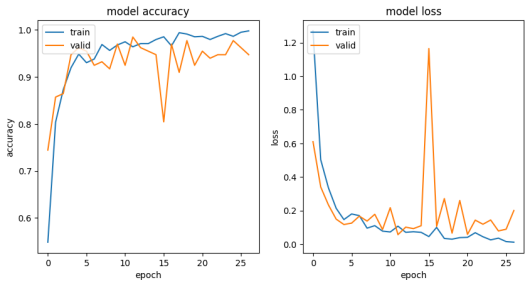
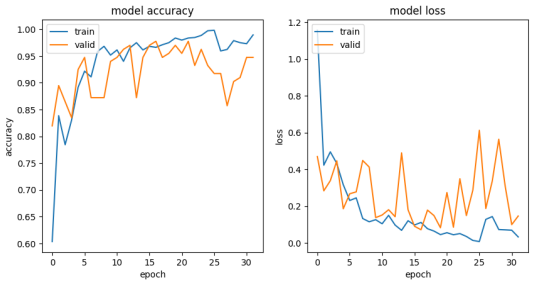


Figura 2 – Experiment 2

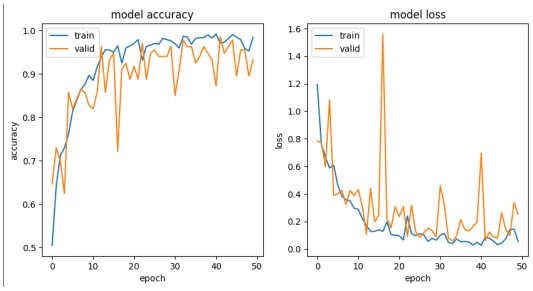
Para os últimos experimentos, nos quais realizamos o Fine-tuning, fica evidente a melhora em relação aos modelos anteriores. Amplificamos o escopo de aprendizagem dos modelos em relação ao experimento 1, uma vez que já possuímos pesos pré-treinados na ImageNet, ao mesmo tempo que permitimos que ele aprenda com o novo conjunto de dados, melhorando o experimento 2.



(a) 3-a



(b) 3-b



(c) 3-c

Figura 3 – Loss and Accuracy, Experiment 3

Assim, obtemos resultados muito bons! É importante observar, porém, a proximidade dos resultados 3-a, 3-b e 3-c. A melhora é pequena e oscilante, o que nos mostra que o ganho entre uma e outra é, em geral, pouco relevante. Essa observação é interessante porque, embora os resultados sejam extremamente parecidos, o tempo de treinamento dos modelos é sensivelmente diferente: à medida em que aumentamos o número de parâmetros treináveis a partir da 3-a — em que somente algumas camadas são treinadas —, até a 3-c — em que todas as camadas são treinadas —, os modelos gradativamente levam mais tempo em treinamento. Levando em conta limitações computacionais e temporais, a diferença de tempo pode ser muito grande para pouca melhora em termos de performance.