Trabalho prático 4

Transfer learning

Julianny Favinha Donda
156059
julianny.favinha@gmail.com

José Henrique Ferreira Pinto
155976
joseh.fp@gmail.com

Objetivo

O objetivo do trabalho 4 é entender e aplicar o conceito de *Transfer Learning*. Vamos usar uma rede já treinada para um problema específico para solucionar um outro problema, sem a necessidade de treinar a rede com a totalidade das imagens de treino. O treinamento do zero leva um tempo considerável e é necessário um conjunto de dados muito extenso. A abordagem explorada facilita e acelera a solução do problema proposto, principalmente se ambos conjuntos de dados são similares na estrutura e no conceito.

Atividade

Nessa atividade, foi proposto utilizar a arquitetura da rede *SqueezeNet* pré-treinada no conjunto de dados ImageNet e fazer um *fine-tune* para o conjunto de dados CIFAR-10.

A *SqueezeNet*[5] proposta em 2016, ficou famosa por alcançar resultados comparáveis aos das redes estado da arte na época, mas com 50 vezes menos parâmetros. Essa redução foi alcançada utilizando blocos de convolução chamados *FireModule*. O módulo é composto de dois componentes, o *Squeeze*, filtros 1x1, e o *Expand*, que contém filtros de 1x1 e 3x3. Os filtros de *Squeeze* garantem uma entrada reduzida aos filtros de *Expand* explicando a redução de parâmetros alcançada. Ligando vários desses módulos, e finalizando com uma classificação baseada em RELU e Softmax, a rede classificou com bons resultados a ImageNet, com suas 1000 classes.

Os dois conjuntos (ImageNet e CIFAR-10) se tratam de imagens e, mais especificamente, o conjunto CIFAR-10 classifica imagens em 10 classes, sendo elas “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship” e “truck”.

A proposta do *Transfer Learning* é que a rede base foi treinada e aprendeu a selecionar características importantes das imagens, conforme esta passava por suas camadas. No final, antes da classificação, é entregue um vetor com características condensadas. Substituindo a última (ou últimas) camada(s) conseguimos ler essas características e aplicar nosso próprio classificador.

Vamos testar três modelos diferentes para *Transfer Learning*. Em todos os modelos, as três últimas camadas da SqueezeNet foram substituídas, pois estas são responsáveis pela classificação da ImageNet. Como foi dito anteriormente, a ImageNet classifica em 1000 classes e a CIFAR-10 em apenas 10 classes.

No primeiro modelo, vamos congelar todas as camadas da rede base. Esperamos um resultado não muito bom, por termos apenas alterado o classificador. Depois, vamos descongelar algumas camadas do topo da SqueezeNet. Isso fará com que as camadas mais perto do classificador sejam afetadas por este, realizando o *fine-tuning* dos pesos e melhorando nosso resultado. Esperamos uma melhora na acurácia. Por último, vamos descongelar todas as camadas. Esperamos o melhor resultado nesse experimento.

Experimento e resultados

Parte do código foi disponibilizado pela professora com as instruções para ser completado. Porém, foi necessário entender como implementar os trechos faltantes.

Procuramos referências de como usar as ferramentas recomendadas, principalmente na documentação do Keras[1], onde pudemos avançar bastante.

A primeira coisa a ser feita foi preparar os dados da CIFAR-10. Esse conjunto de dados é separado da seguinte maneira: 50000 observações para o treinamento e 10000 observações para teste. Separamos também o conjunto de treinamento em 40000 para treino e 10000 para validação.

Além disso, foi necessário fazer o pré-processamento dos *targets*, para conformar com a saída da rede, que agora tem 10 possíveis classificações. Isso foi feito usando a técnica de converter um vetor de inteiros em uma matriz binária[3].

Os modelos foram executados na primeira vez usando um *learning rate* igual 0,0001 e 10 épocas. Outros experimentos envolveram diminuir o learning rate e aumentar o número de épocas. O otimizador escolhido foi o Adam e a *loss* a *Categorical Cross entropy*.

SqueezeNet com todas as camadas congeladas

```
10000/10000 [==============================] - 16s 2ms/step
SqueezeNet with frozen layers
Validation loss: 2.2694514072418213
Validation accuracy (NORMALIZED): 0.1312
```

Figura 1 - Trecho de captura de tela para o algoritmo *SqueezeNet* com todas as camadas congeladas, obtendo loss de 2,27 e acurácia 13,1% para o conjunto de validação, *learning rate* igual a 0,0001 e 10 épocas.

Notamos que diminuir o *learning rate* não melhorou os resultados. Ao executar o algoritmo por 20 épocas, a acurácia também dobrou. Como esperado, a acurácia foi bastante ruim, pois só alteramos o classificador.

SqueezeNet treinando os dois últimos *Fire Modules*

```
10000/10000 [==============================] - 14s 1ms/step
Training last 2 Fire Modules + classification layers
Validation loss: 1.4662954704284668
Validation accuracy (NORMALIZED): 0.4942
```

Figura 2 - Trecho de captura de tela para o algoritmo SqueezeNet com algumas camadas congeladas, obtendo loss de 1,47 e acurácia 49,4% para o conjunto de validação, *learning rate* igual a 0,0001 e 10 épocas.

Notamos que diminuir o *learning rate* para 0,000001 não melhorou os resultados. Ao executar o algoritmo por 20 épocas, a acurácia aumentou para 51,3%. Agora que temos as últimas camadas sendo alteradas para se adequar ao nosso novo classificador, notamos uma melhora da acurácia.

SqueezeNet com *fine-tuning* de todas as camadas

```
10000/10000 [==============================] - 5s 482us/step
Fine-tuning all layers
Validation loss: 0.8880291116714477
Validation accuracy (NORMALIZED): 0.7243
```

Figura 3 - Trecho de captura de tela para o algoritmo SqueezeNet com nenhuma camada congelada, obtendo loss de 0,89 e acurácia 72,4% para o conjunto de validação, *learning rate* igual a 0,0001 e 10 épocas.

Notamos que diminuir o *learning rate* para 0,000001 não melhorou os resultados. Ao executar o algoritmo por 20 épocas, a acurácia também não melhorou significativamente.

Porém, como suposto, temos a melhor acurácia. Isso pode ser atribuído a todos as camadas poderem agora se adaptar ao novo classificador, conseguindo ajustar de forma mais precisa os resultados.

Tensorboard

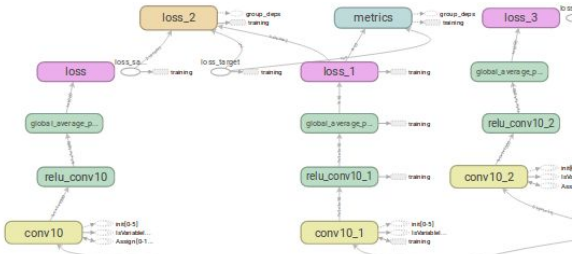

Figura 4 - Captura de tela de trecho do grafo exibido no Tensorboard para um determinado modelo. Nele, é possível ver as camadas da rede e como estão conectadas, sendo mais fácil a visualização.

Conclusões e trabalho futuro

Dos resultados anteriores, pudemos observar que o melhor modelo é aquele que não congelou nenhuma camada da SqueezeNet.

Usamos o Adam como otimizador para o modelo. Como trabalho futuro, poderíamos ter testado o mesmo algoritmo para outros otimizadores, como SGD e Adagrad[4].

Também poderíamos ter executado os modelos por mais épocas, verificando se o resultado na acurácia ou loss melhorava.

Outras coisas que poderíamos ter melhorado é o tempo de execução dos modelos. Notamos que quanto mais tempo deixamos executando, melhores os resultados. O ideal seria encontrar um número de épocas que não propagasse *overfit*.

Referências

[1] **Model class API from Keras Documentation.** Disponível em <https://keras.io/models/model/> Acesso em 18 nov 2018.

[2] **How to use models from keras.applications for transfer learning.** Disponível em <goo.gl/1aASXA> Acesso em 18 nov 2018.

[3] **Keras utils to_categorical.** Disponível em <https://keras.io/utils/#to_categorical> Acesso em 19 nov 2018.

[4] **Keras Optimizers.** Disponível em <https://keras.io/optimizers/> Acesso em 19 nov 2018.

[5] **Deep Learning Reading Group: SqueezeNet.** Disponível em <https://www.kdnuggets.com/2016/09/deep-learning-reading-group-squeezenet.html> Acesso em 22 nov 2018.