Projeto Final

Nanodegree – Engenheiro de Machine Learning

Rafael Araújo
23 de setembro de 2017

Definição

Sobre o Projeto

O projeto consiste em construir uma plataforma social de descrição de imagem. Os usuários podem enviar imagens que desejam, assim outros usuários podem dar suas descrições para essas imagens. De tempos em tempos, o sistema aprende a associar palavras a imagens e pode criar novas descrições para elas.

Descrição do Problema

Criar um sistema onde usuários podem descrever e enviar imagens, depende apenas de criar uma plataforma onde essa interação possa ser feita, isto é, uma aplicação onde o usuário possa enviar imagens e possa descreve-las. Uma aplicação como essa para um dispositivo móvel, como por exemplo, um aparelho Android, depende de um aplicativo Android para que os usuários possam interagir com o sistema. Nessa etapa, o problema se define em construir uma aplicação Android onde os usuários possam enviar essas imagens e descrevê-las. Isso pode ser feito com uma aplicação Android e com um sistema de backend, um sistema anterior ao aplicativo, que fornece serviços REST (Representational State Transfer), onde, por meio de chamadas HTTP, pode interagir com a forma de armazenamento das imagens e das descrições, além de interagir com qualquer operação que esse sistema venha a implementar.

O requisito do projeto nos pede para implementar uma forma onde o sistema aprende a associar imagens a palavras, podendo criar novas descrições automaticamente. Segundo Vinyals et al. (2015), a descrição automática do conteúdo de uma imagem é um problema fundamental na inteligência artificial que envolve visão computação e processamento de linguagem natural. Assim, para atender ao requisito, deverá ser construído um modelo de inteligência artificial que possa criar essa associação entre imagem e palavra.

Construindo o modelo acima, podemos treiná-lo em alguma base de dados de imagens e descrições, permitindo ajustes para que a performance seja ideal e, em seguida, podemos usar o mesmo modelo para ser treinado sobre a base de dados formada pelo sistema no qual a aplicação Android se comunica, gerando assim, descrições automáticas para essa aplicação.

Uma ideia de modelo que pode relacionar imagens a palavras seria unir modelos que processam bem imagens, como os modelos de convolução (convolutional neural networks - CNN), com modelos recorrentes (recurrent neural networks - RNN) que tem boa performance para geração de linguagem natural. Essa proposta foi dada por Vinyals et al. (2015) e é usada como solução para esse tipo de problema.

É esperado que, após o modelo ser construído e treinado, para cada imagem, o modelo seja capaz de gerar uma descrição suficientemente boa para aquela imagem. Assim esse modelo pode ser utilizado pela aplicação Android.

Métricas

De modo a avaliar o modelo proposto, usaremos o método BLEU, Bilingual Evaluation Understudy (Papineli et al. 2002). É um método de avaliação de tradução de máquina que é rápido, barato e independente de linguagem, que correlaciona muito bem avaliações humanas (Papineli et al. 2002). Um sistema de tradução gera uma nova sentença a partir de uma sentença em um outro idioma. No nosso caso, geramos uma sentença a partir de uma imagem, o que torna a métrica útil no caso. O método BLEU é utilizado em diversos trabalhos na área de descrição de imagens, como podemos ver em Vinyals et al. (2015) e Chen et al. (2014), para a avaliação das descrições geradas pelos modelos.

Para calcular o índice BLEU, são necessários uma referência e um candidato. A ideia é analisar o quão próximo da referência, o candidato se parece. O índice BLEU utiliza em seu cálculo dois conceitos importantes: a precisão modificada de n-grama e a penalidade por brevidade de sentença. A precisão modificada de n-grama é uma razão sobre a quantidade de palavras em uma referência. A penalidade por brevidade de sentença garante que um candidato a tradução com escore alto estará de acordo com o tamanho da referência, com a escolha de palavras e com a ordem (Papineli et al., 2002). A seguir, o índice BLEU é calculado utilizando uma média geométrica dos escores de precisão modificada do texto e multiplicado pela exponencial da penalidade de brevidade. O desenvolvimento da ideia e a avaliação do índice pode ser encontrada em detalhes no trabalho de Papineli et al. (2002).

Análise

Exploração de Dados

O projeto possui, como entrada de dados, as imagens e descrições fornecidas pelos usuários. Para que isso seja feito, uma aplicação Android foi construída, que, utilizando um WebService, grava os dados no servidor.

Aplicação Android e Entrada de Dados

O objetivo do projeto é criar uma plataforma social de descrição, que podemos acessar de um aparelho Android. Para isso, foi criado o SocialPicture, um aplicativo que permite enviar imagens, enviar descrições e ver imagens e descrições de outros usuários. O aplicativo funciona da seguinte forma: um usuário envia uma imagem, que fica disponível para todos os outros usuários. Qualquer imagem presente no “feed” de imagens, isto é, na lista contendo todas as imagens enviadas, podem ter descrições adicionadas. De tempos em tempos, o sistema gera uma descrição para a imagem e mostra para todos os usuários essa descrição.

Ao abrir a aplicação, o usuário é apresentado a uma lista de imagens enviadas pelos usuários. Nessa tela, é possível atualizar a lista, clicar no botão “+” para adicionar uma nova imagem ou adicionar uma descrição clicando sobre a imagem.

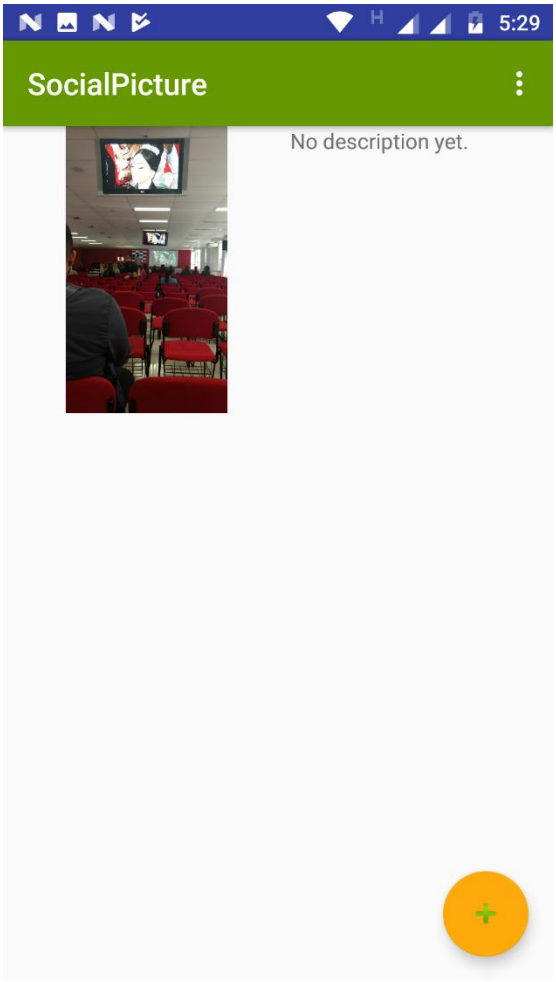

Figura 1 - Lista de imagens

Ao adicionar uma nova imagem, é possível selecionar uma imagem da galeria ou tirar uma foto pela câmera do dispositivo. Depois, basta salvar a imagem no servidor.

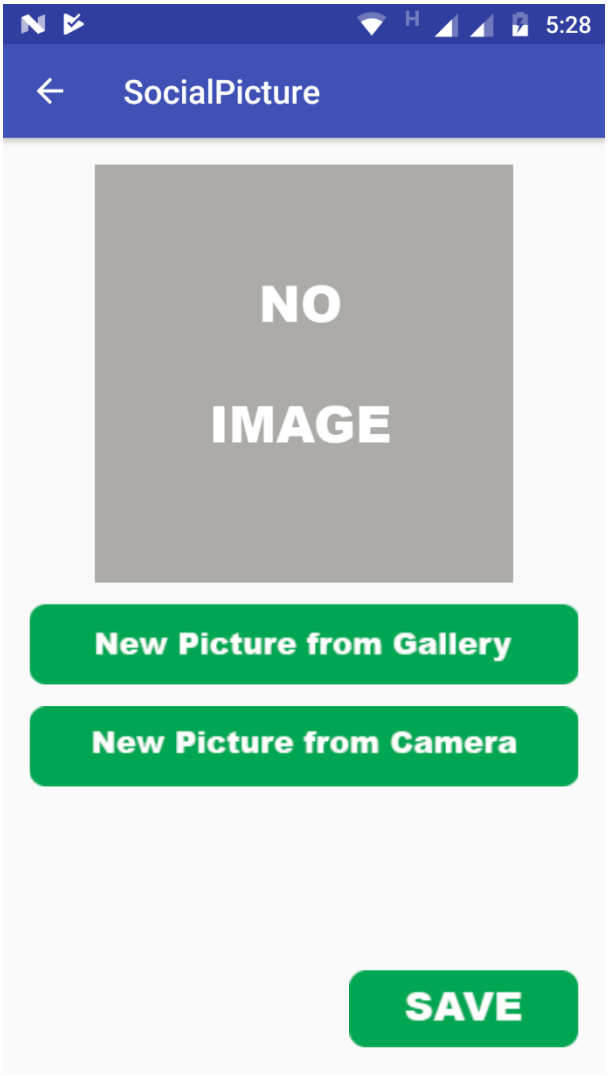

Figura 2 - Adicionar Imagem

Ao clicar em uma imagem, é possível adicionar uma descrição para a imagem. Basta digitar a nova descrição e clicar em salvar.

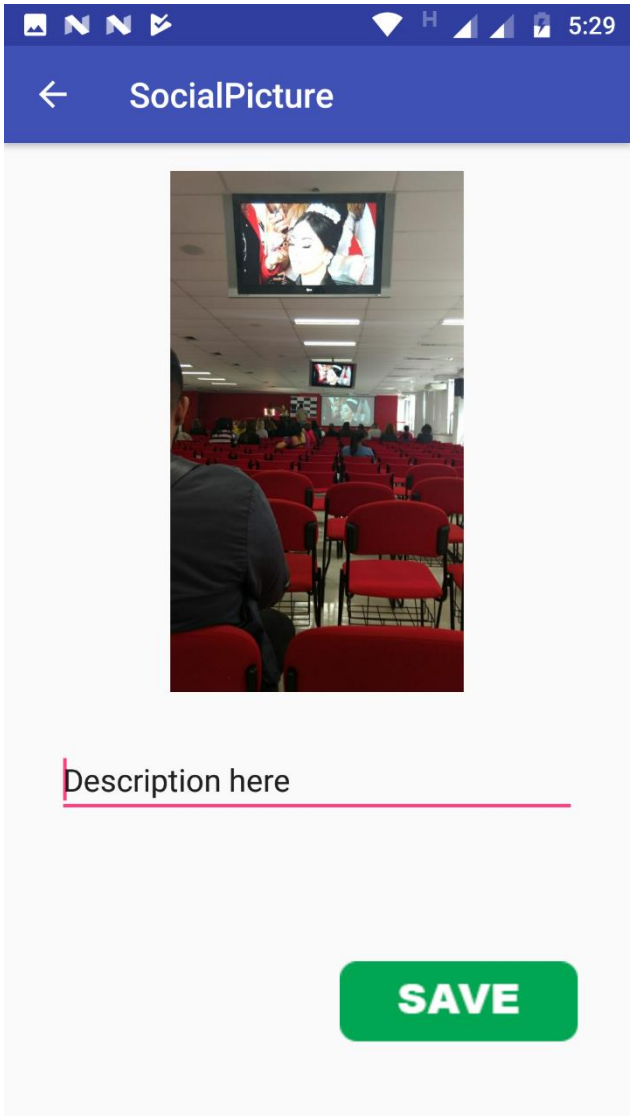

Figura 3 - Inserir descrição

Por trás da aplicação Android, existe um servidor em Python que recebe as imagens e as descrições. O servidor é um WebService REST, acessível por HTTP e tem duas funcionalidades. A primeira é salvar e listar as imagens, e salvar e mostrar descrições. A segunda funcionalidade é executar o modelo e gerar novas descrições. A segunda funcionalidade será explicada mais adiante.

As imagens e descrições são salvas diretamente no servidor, não necessitando de qualquer sistema de banco de dados.

Dados de teste do modelo

Para avaliar modelos de descrição de imagens, existem diversos conjuntos de dados disponíveis, sendo os mais comuns, o MS COCO, o maior dos conjuntos de dados, o Flickr-30k e o Flickr-8k.

Foi utilizado, para testar o modelo, o conjunto de dados Flickr-8k (https://illinois.edu/fb/sec/1713398). O conjunto apresenta cerca de 8000 imagens da plataforma Flickr e descrições dadas por usuários da plataforma para cada imagem, sendo cerca de 6000 imagens e descrições para treinamento, 1000 para validação e 1000 para teste. Os dados se encontram no conjunto de dados, formatados por tabulação e contém um diretório de imagens. Cada registro no arquivo formatado contém cada descrição para cada imagem e identifica o arquivo da respectiva imagem no diretório.

Algoritmos e Técnicas

O modelo utilizado foi construído baseando-se no modelo proposto por Vinyals et al. (2015). O código foi baseado na construção realizada por Mishra, Salvador e por Arriaga.

O princípio aplicado ao modelo é de considerar um modelo de tradução de linguagens e torna-lo um modelo onde é possível “traduzir” a imagem em linguagem. Segundo Vinyals et al. (2015), trabalhos recentes demonstraram que a tarefa de tradução pode ser alcançada de maneiras mais simples utilizando-se redes neurais recorrentes (RNNs). Esses modelos exibem um codificador RNN, para a entrada de sentenças a serem traduzidas e um decodificador, para gerar a sentença traduzida. A ideia é substituir o codificador por uma rede de convolução (CNN). Nos últimos anos, é visível que as CNNs podem produzir uma rica representação da imagem de entrada combinando-a com um vetor de tamanho fixo, sendo essa representação utilizada em diversas tarefas no campo da visão computacional (Vinyals et al. 2015).

Podemos ver diversos trabalhos no campo da visão computacional que utilizam tais redes, como o modelo VGG-16 de Simonyam e Zisserman (2015) e a conhecida rede GoogLEnet de Szegedy et al. (2014), que implementou a arquitetura Inception. Ambos os trabalhos se utilizam de CNNs para classificar imagens.

No modelo proposto, todas as descrições são utilizadas para criar um vocabulário de palavras que o modelo pode gerar, o que permite que cada sentença seja codificada em uma lista de índices referenciando palavras no vocabulário. As imagens são codificadas por uma CNN que envia os dados para o modelo que se utiliza de RNNs para gerar sentenças de descrições.

O modelo foi construído da seguinte forma: como codificador de imagem, foi utilizado um modelo VGG-16 completo, utilizando-se os pesos atribuídos para o conjunto de dados ImageNet, sendo que a saída são as probabilidades de a imagem pertencer a alguma das 1000 classes do classificador. Segundo Vinyals et al. (2014), um modelo CNN deve ser usado e a saída deve ser coletada da última camada escondida. Essa abordagem foi escolhida para minimizar a quantidade de dados processados devido a limitação de hardware para o projeto. A implementação feita por Mishra utiliza a mesma abordagem e conseguiu com sucesso gerar descrições concretas para as imagens. Foram realizados testes usando a última cada densa da rede VGG-16 e o resultado foi semelhante ao apresentado com o modelo proposto. Na figura 4, há um diagrama do modelo e suas camadas. Como otimizador, foi escolhido o algoritmo RMSProp por ter o melhor desempenho. O algoritmo adapta a taxa de aprendizado de acordo com cada parâmetro do modelo. Chega-se a um resultado divindindo a taxa de aprendizado por um peso, tomando médias das magnitudes dos gradientes recentes deste peso.

Para a tarefa de codificação de imagens, diversos modelos podem ser usados, como, por exemplo, modelos Inception V3 (Szegedy et al., 2015), VGG-16 (Simonyam & Zisserman, 2015), ResNet (He et al., 2015), entre outros. Para se adequar ao hardware limitado, foi escolhido o modelo VGG-16 devido a sua boa performance em tarefas de classificação e por ser ideal em hardwares com memória limitada. O modelo VGG-16 garantiu primeiro e segundo lugar no desafio ImageNet ILSVRC-2014, tendo como erro de validação, 7,4% sobre a base de teste ImageNet. O modelo consiste em

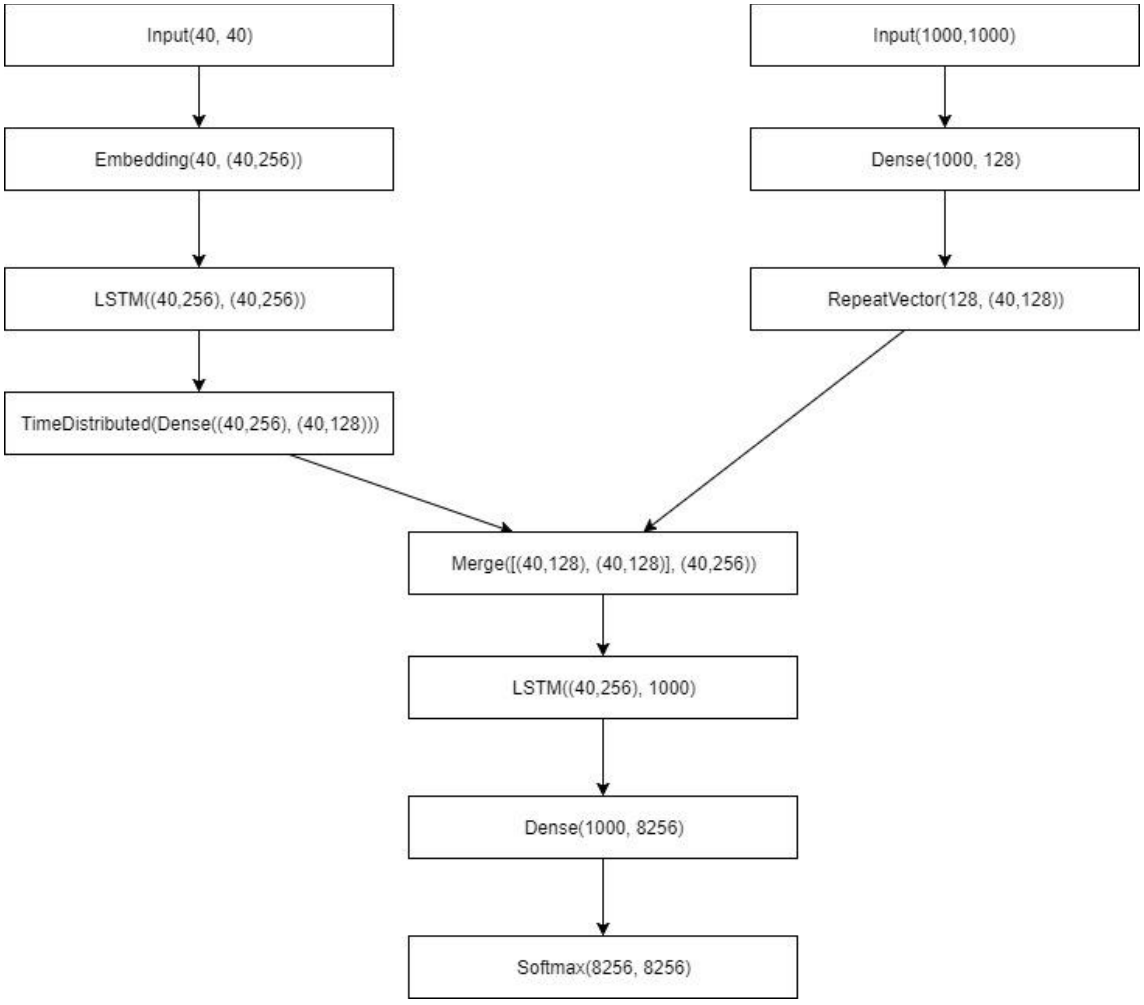

Figura 4 - Modelo de rede neural para descrição automática de imagens

13 convoluções com filtro 3x3 e 3 camadas densas e visa classificar uma imagem em 1000 classes diferentes.

Após a codificação, o modelo final recebe duas entradas, sendo a primeira, uma lista de índices de cada palavra da sentença em relação ao vocabulário. A segunda entrada é a lista de probabilidades de a imagem pertencer a uma entre 1000 classes. É aplicado um tipo de RNN chamada LSTM (Long Short Term Memory). Segundo Vinyals et al. (2014), a rede LSTM é o estado da arte em tarefas de tradução. Existe outro tipo de RNN que pode ser utilizada, a GRU (Gated Recurrent Unit). Ambas têm o mesmo efeito no modelo, portanto foi escolhida a LSTM a fim de manter consistência ao trabalho original de Vinyals et al. (2015).

Benchmark

Usaremos como benchmark, os dados obtidos por Mishra em seu estudo de código para um gerador de descrição. Em seu estudo, foi criado um modelo muito semelhante ao utilizado aqui, obtendo perda de aproximadamente 2,65 em 50 épocas de treinamento, assim como índice BLEU de aproximadamente 0,57. Por mais que o benchmark não esteja em um trabalho mais detalhado como o de Vinyals et al. (2015), este é um bom benchmark pois o mesmo código utiliza-se do mesmo conjunto de dados, bibliotecas e frameworks utilizados neste projeto. Além disso, no trabalho de Vinyals et al. (2015), apenas a ideia do modelo é proposta, cabendo ao leitor a sua implementação. Optamos aqui a seguir a implementação sugerida pela documentação do framework Keras para geradores de descrição, o que nos levou a basear a implementação na solução proposta por Mishra, além de utilizar ideias da solução de Salvador e da solução de Arriaga, todas baseadas no trabalho de Vinyals et al. (2015), com exceção de Salvador que se baseou no trabalho de Xu et al. (2015), uma modificação do trabalho de Vinyals et al. (2015).

Metodologia

Processamento de Dados

Do conjunto Flickr8k, primeiro tratamos dos dados relativos às descrições, que estão em um arquivo texto formatado por tabulação. Cada registro se encontra no seguinte formato:

1305564994_00513f9a5b.jpg#0 A man in street racer armor be examine the tire of another racer 's motorbike .

A primeira parte indica o arquivo da imagem e o número da descrição. Cada imagem possui 5 descrições no arquivo. Assim, foi necessário particionar a linha por tabulação e depois particionar a primeira parte usando o caractere “#”, assim é possível gerar um par arquivo descrição para cada sentença de descrição.

Para as imagens, o único tratamento realizado foi transformar a imagem em um “array” de valores, em seguida foi aplicado um pré-processamento que converte a imagem de RGB (red green blue) para BGR (blue green red), ideal para o processamento pelo VGG-16. Todo o pré-processamento pode ser visto no Notebook Data Preparation Flickr8k.ipynb, no diretório Python.

Implementação

O projeto foi dividido em três etapas: aplicação Android, servidor de WebServices, modelo de rede neural. A aplicação Android foi desenvolvida de forma nativa, utilizando a SDK oficial do Android. O servidor de WebServices foi construído utilizando Python e o framework Flask. Para que a aplicação funcionasse, foi necessário construir formas de acessar todas as imagens e suas descrições geradas pelo sistema, acessar apenas os dados de uma imagem, adicionar imagem e adicionar descrições. Além disso, foi necessário decidir uma forma de codificar os dados para que ambos os lados, aplicação e servidor, entendessem as mensagens trocadas. O protocolo REST define como forma de comunicação, o JSON (Javascript Object Notation), o que nos

ajuda na hora de decidir essa forma de codificação. Por exemplo, a codificação de uma imagem é representada da seguinte forma:

```
{
        "image": "images/1.jpg",
        "id": 1,
        "extension": "jpg",
        "description": "a description"
}
```

Dessa forma, é possível enviar e receber dados sobre as imagens do aplicativo ao servidor sem perda de conteúdo. O servidor gera os dados baseando-se no conteúdo do diretório images e do sistema de geração de descrições, que é baseado no modelo de rede neural criado. Sendo assim, a lista de imagens do servidor é gerada assim que o servidor é iniciado. O servidor permite também que a aplicação leia diretamente do diretório de imagens, permitindo que enviemos apenas o caminho da imagem via JSON e não a imagem propriamente dita.

O servidor ainda conta com a geração automática de descrições que utiliza o modelo de rede neural. O sistema funciona da seguinte forma: de tempos em tempos, uma rotina lê todas as imagens e descrições gravadas e treina o modelo com esses dados. Em seguida, as mesmas imagens passam pelo modelo para gerar uma nova descrição. A descrição então é gravada no objeto de imagens e retornada na lista ou via acesso direto à imagem pelo WebService. Falaremos mais sobre esse processo mais à frente.

O modelo foi construído usando a linguagem Python através da biblioteca Keras. A biblioteca Keras é um facilitador para lidar com os frameworks de redes neurais Theano e TensorFlow. Aqui, utilizamos o TensorFlow como backend da biblioteca. O modelo consiste em duas partes de código, o gerador e o modelo. O gerador é uma classe Python que gera pares de imagens e descrições em lotes (batches), para serem consumidos pelo modelo. Alguns passos são executados antes que o par seja gerado, já que existe um pré-processamento. Ao iniciar a classe gerador, é criado um vocabulário contendo todas as palavras contidas nas descrições, sem repetições. É criado, do vocabulário, um dicionário de “palavras índice”. Vinyals et al. (2015) descreve que, para gerar uma nova descrição, é necessário fornecer parte da descrição. Em seu trabalho, para gerar uma nova descrição, é fornecido apenas o indicador de início de sentença, adicionado a cada descrição. Para treinar o modelo, fornecemos toda a sentença exceto pela última palavra, de modo que o modelo gere apenas a última palavra da sentença. Para comparação, é fornecida a última palavra. Assim, o gerador codifica a sentença, excluindo a última palavra utilizando o dicionário de palavra índice e faz o mesmo com a última palavra. Assim, a tripla “imagem, sentença parcial e próxima palavra” é retornada em lote para o modelo. O gerador pode ser encontrado em Python/picturedata/generator.py.

O modelo foi construído em um notebook e implementa todas as camadas descritas na sessão Algoritmos e Técnicas. O modelo executa 30 épocas usando o RMSProp como otimizador e, ao final, tanto o modelo quanto seus pesos são salvos para uso futuro. O modelo pode ser encontrado em Python/Image Captioning V3.ipynb. A avaliação do modelo é feita no notebook Python/Image Caption Evaluation Test Data.ipynb. Nesse notebook, o modelo já treinado é carregado, assim como as imagens codificadas e as descrições. Agora, apenas o identificador de início de sentença é codificado pelo dicionário “palavra índice” e enviado ao modelo, junto da imagem codificada. O modelo então retorna uma lista de probabilidades, onde o índice é o mesmo do vocabulário. Quanto maior a probabilidade, maior a precedência da palavra, isto é, quanto maior é a chance de ela aparecer na sentença, maior a chance de ela

estar à frente na sentença. O modelo então ordena a lista e gera a sentença até que o indicador de fim de sentença seja encontrado. Após isso, a lista é decodificada usando a lista de vocabulário.

Todos os códigos de teste e criação do modelo foram executados em uma máquina Amazon EC2 do tipo p2.xlarge, que possui uma GPU NVIDIA Tesla K80 com 16GB de RAM. Todos as execuções de redes neurais executaram em processamento GPU que garantiu uma redução no tempo de treinamento do modelo.

Para se integrar com a aplicação Android, o código do modelo e códigos adicionais foram transferidos para o servidor da aplicação, criando o sistema de geração de descrição automático. O sistema executa os mesmos passos do processo executado para o modelo de teste, com algumas modificações. Primeiramente as descrições são lidas do arquivo e é gerado uma tabela contendo a descrição e o arquivo de imagem correspondente àquela descrição. A seguir, cada imagem relacionada na tabela é processada pelo VGG-16. Após isso, o modelo é criado novamente, já que não podemos aproveitar o modelo de teste salvo devido aos parâmetros diferentes, isto é, o tamanho máximo da descrição é diferente se considerarmos todo o conjunto de dados Flickr8k e nosso sistema que varia a todo momento. Não podemos utilizar o modelo já treinado pois o requisito do sistema nos diz que precisamos utilizar as descrições enviadas pelos usuários no treinamento do modelo, assim novos treinamentos serão requeridos, o que invalida o uso do modelo treinado por ter tamanho de descrição máximo diferente. Durante o treinamento, o gerador deve transmitir os dados de um a um e não a cada 32 como foi configurado para o modelo de testes. Isso permite que um novo registro no aplicativo seja suficiente para que o processamento seja feito.

Para que esses passos fossem concluídos, um novo gerador foi criado, o CaptionGen.py, que contém todos os passos, seja de pré-processamento, treinamento, gerador de dados para treinamento e gerador de descrição. Toda vez que é solicitada uma nova geração de descrição, todos os passos no gerador são executados novamente, isso garante que um novo vocabulário é gerado, contendo um novo valor para o tamanho máximo de descrição, assim gerando um novo modelo. Isso garante que todas as amostras enviadas pelo aplicativo serão processadas com base nas descrições fornecidas pelas próprias amostras. Para que o processo fosse executado novamente, após a inicialização do servidor e após um tempo determinado, foi criado um processo latente que executa a cada período de tempo. Esse período foi configurado inicialmente para 2 horas, já que o processo de treinamento pode ser longo se a quantidade de dados para processamento for grande, o que pode enfileirar o processo e ocasionar erros.

As descrições geradas foram medidas de acordo com o índice BLEU. Utilizamos a biblioteca NLTK para Python para utilizar o índice em nossos códigos.

Refinamento

Inicialmente, o modelo utilizava como otimizador, o algoritmo SGD (Stochastic Gradient Descent), executava 50 épocas com 120 passos por época. A perda era de aproximadamente 5,54. Começamos o refinamento alterando o número de passos por época, já que seria necessário mais passos para que, a cada época, o modelo percorresse todo o conjunto de dados. O cálculo para número de passos é dado pelo número de amostras divido pelo tamanho do batch. Houve melhora significativa após esse ajuste, com a perda caindo para 4,98. Foi notado subajuste (underfitting) durante o treinamento, com o aumento da perda ao longo das épocas. Notamos também que, em média, a menor perda se situava entre as épocas 1 e 30, assim, reduzimos para 30 épocas de treinamento, pois não havia benefícios para treinar mais épocas além de 30. Com isso, passamos a utilizar os pesos da melhor época e não os pesos da última época, pois os mesmos se mostravam ruins, com perdas muito altas.

Outra mudança realizada foi a de alterar o otimizador para o algoritmo RMSProp. Foram testados diversos algoritmos, sendo o RMSProp o melhor. A perda melhorou, caindo apenas para 3,22.

Resultados

Avaliação e Validação do Modelo

O modelo foi treinado em 30 épocas e obteve perda de aproximadamente 3,22. O modelo foi testado usando o grupo de testes do conjunto de dados Flickr8k com aproximadamente 1000 imagens. O modelo conseguiu aproximar a descrição de algumas imagens mas cometeu muitos erros, além disso algumas estruturas da sentença estão fora do padrão da língua inglesa. Nota-se também um vocabulário pobre nas descrições, evidenciando que o modelo não considerou todo o vocabulário gerado pelas descrições de treinamento. Tais comportamentos estão relacionados a alta perda do modelo. Mesmo diante dessa perda, o fato do modelo demonstrar capacidade de gerar descrições parcialmente alinhadas com as imagens, mostra o quão forte é a arquitetura dele. Tal perda alta se deve à falta de afinamento do modelo, que deve ser feito na camada CNN e também na camada RNN e por fatores ambientais como hardware. Podemos ver nas figuras 5, 6 e 7 um exemplo de imagem corretamente descrita, uma com descrição parcial e uma imagem descrita incorretamente. Mesmo diante a esse problema, obtemos um índice BLEU bastante promissor, de aproximadamente 0,50.

O modelo atende ao problema pois é devidamente capaz de gerar descrições relativas às imagens, mesmo que, neste caso, não tenha gerado descrições eficientes. É confiável pois sempre vai gerar uma descrição similar a cada fase de treinamento dado as mesmas circunstâncias como tamanho máximo de descrição e número total de amostras.

A black and white dog is in the grass.

Figura 5

A man in a black shirt is in a field.

Figura 6

A little girl in a black shirt
is in the background.

Figura 7

Ao utilizar o modelo no servidor WebService para a aplicação Android, notamos que é possível utilizar o mesmo modelo para tal tarefa, porém, a aplicação gera uma quantidade pequena de dados, o que não favorece o treinamento do modelo. A cada treinamento, com poucos dados, descrições sem sentido são geradas, o que significa que são geradas, não só descrições incompatíveis com a imagem como descrições não coerentes com a língua inglesa. A medida que a aplicação coleta grande quantidade de dados, o modelo passa a ser eficiente.

Justificativa

Optamos por utilizar os valores de perda e índice BLEU alcançados por Mishra como referência. Após nossas análises chegamos a uma perda de 3,22 que logo sobe para acima de 4 durante o treinamento. Por mais que tentássemos otimizar o modelo, não foi possível diminuir a perda para menos que esse valor, sendo esse valor alcançado apenas uma vez em muitos treinamentos. Quanto ao índice BLEU, atingimos aproximadamente 0,50, em comparação aos 0,57 do estudo de Mishra. Um índice bem promissor. De todo modo, diante de todos os aspectos discutidos acerca do modelo e como este se comportou, não consideramos o modelo, da forma que está, ideal para a solução do problema, mas consideramos o modelo como promissor para a solução do problema. O modelo não conseguiu gerar descrições suficientemente boas para as imagens do grupo de teste do conjunto de dados. Além disso, é praticamente impossível que o modelo seja utilizável em um aplicativo Android onde o fornecimento de imagens é lento. Um modelo de rede neural precisa de milhares de amostras para ter um bom funcionamento, não gerando nenhum resultado útil para 10 ou 20 amostras.

Este modelo, com algumas customizações, pode ser usado para a solução do problema, pois evidenciamos que ele é capaz de gerar descrições corretas se for devidamente ajustado. O trabalho de ajuste é complexo e não foi considerado devido aos custos de processamento demandados pelo hardware.

Conclusão

Reflexão

O problema proposto era criar uma plataforma onde não só usuários pudessem descrever imagens que eles mesmo tirassem como a própria plataforma pudesse também descrever automaticamente essas imagens. Para resolver esse problema, criamos uma aplicação Android que pudesse salvar imagens, salvar descrições e listar essas imagens e descrições geradas pelo sistema. Acompanhando essa aplicação, criamos um servidor que contém um WebService que permite salvar imagens e descrições assim como gerar descrições utilizando um modelo de rede neural. Essa rede neural foi criada a partir de uma rede convolucional (CNN) e uma rede recorrente (RNN), que juntas, conseguiram gerar descrições para imagens em um conjunto de dados de imagens do Flickr chamado Flickr8k.

Por mais que o modelo criado não tenha alcançado o desempenho esperado, é interessante notar como a união de redes tão diferentes como a CNN e a RNN conseguem de forma tão consistente, gerar descrições em imagens. Nosso modelo, conseguiu gerar descrições corretas para algumas imagens, o que demonstrou o poder que o modelo tem. Outras descrições foram parciais, não descrevendo a cena como todo, mas descrevendo algum detalhe da cena, algo também interessante e comum nas redes CNN que são excelentes em detectar detalhes em imagens.

Das dificuldades encontradas, a maior foi tentar melhorar a performance do modelo. Um modelo, às vezes, chega a um dado momento em que sua performance não melhora sem um ajuste fino, detalhado, nas suas camadas. Esse ajuste é complexo e leva tempo, pois um ajuste errado levaria em uma situação de sobreajuste (overfitting). Ajustes mais simples como a troca do otimizador, mudança nos passos por época e outros parâmetros podem ajudar, mas não tem tanta influência quanto a um bom ajuste fino.

O modelo é o correto para esse tipo de problema, mas precisa de inúmeros ajustes antes que este possa ser usado como solução. Vimos que o modelo consegue gerar descrições, mas não da melhor forma e nem para todas as imagens ou para a maioria, mas o mesmo modelo pode ser ajustado para tal fim.

Aprimoramento

Para que o modelo possa ser usado, este precisa de ajustes finos nas camadas da rede. Existem diversos ajustes finos que devem ser feitos em redes CNN, como adicionar novas camadas densas ou retirar camadas. Para redes RNN, existem diversos artigos sobre o assunto, como o trabalho de MacNeil e Eliasmith (2011). Esse tipo de ajuste complexo pode gerar um modelo mais complexo, mas que tenha melhor performance sobre este modelo. Outra mudança é adaptar o modelo para o trabalho de Xu et al. (2015) que propõe modelo de descrição de imagens com atenção espacial.

Além das mudanças no modelo, a forma com que o modelo é utilizado na aplicação também precisa ser melhorada. No estágio atual, o modelo não consegue gerar nada se a quantidade de imagens enviadas for baixa. Talvez se esquecermos a necessidade de utilizar as descrições dos usuários no treinamento e utilizar um modelo pré-treinado, este problema possa ser resolvido.

Aqui, utilizamos como base, o trabalho de Vinyals et al. (2015), mas outros trabalhos vieram antes que podem ser usados para resolver o mesmo problema. O trabalho de Vinyals et al. (2015) é o mais comum na área de descrição de imagens, mas outros também tem seus trabalhos publicados. Assim, existem diversas outras soluções para esse problema.

Bibliografia

- Vinyals, Oriol. Toshev, Alexander. Bengio, Samy. Erhan, Dumitru. Show and Tell: An Image Caption Generation. Google, 2015.
- Papineli, Kishore. Roukos, Salim. Ward, Todd. Zhu, Wei-Jing. BLEU: A Method for Automatic Evaluation of Machine Translation. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, 2012.
- Chen, Xinlei. Zitnick, Lawrence. Learning a Visual Representation for Image Caption Generation. IEEE Conference on Computer Vision and Pattern Recognition, 2015.
- Mishra, Anurag. An Image Captioning Project. https://github.com/anuragmishracse/caption_generator, 2017.
- Arriaga, Octavio. Neural Image Captioning. https://github.com/oarriaga/neural_image_captioning, 2017.
- Salvador, Amaia. Image Captioning with Spatial Attention in Keras. https://github.com/amaiasalvador/imcap_keras, 2017.
- Simonyan, Karen. Zisserman, Andrew. Very Deep Convolutional Networks for Large Scale Recognition. ICLR, 2015.
- Szegedy, Christian. Liu, Wei. Jia, Yangqing. Sermanet, Pierre. Reed, Scott. Aguelov, Dragomir. Erhan, Dumitru. Vanhoucke, Vincent. Rabinovich, Andrew. Going Deeper with Convolutins. Google, 2014.
- Szegedy, Christian, Vanhoucke, Vincent. Ioffe, Sergey. Shlens, Jonathon. Rethinking Inception Architecture for Computer Vision. Google, 2015.
- He, Kaiming. Zhang, Xiangyu. Ren, Shaoqing. Sun, Jian. Deep Residual Learning for Image Recognition. Microsoft, 2015.
- Xu, Kelvin. Ba, Jimmy. Kiros, Ryan. Cho, Kyunghyun. Courville, Aaron. Salakhutdinov, Ruslan. Zemel, Richard. Bengio, Yoshua. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.
- MacNeil, David. Eliasmith, Chris. Fine-Tuning and the Stability of Recurrent Neural Networks. PLoS ONE, 2011.