TensorFlow's codes - dcgan, classification and cnn

1. DCGAN

Este tutorial demonstra como gerar imagens de dígitos manuscritos usando uma Deep Convolutional Generative Adversarial Network (DCGAN). O código é escrito usando a API Keras Sequential.

Nas Generative Adversarial Networks (GANs) dois modelos são treinados simultaneamente por um processo adversário. Um *gerador* ("o artista") aprende a criar imagens que parecem reais, enquanto um *discriminador* ("o crítico de arte") aprende a distinguir imagens reais de falsificações. Durante o treinamento, o *gerador* se torna progressivamente melhor em criar imagens que parecem reais, enquanto o *discriminador* se torna melhor em diferenciá-las. O processo atinge o equilíbrio quando o *discriminador* não consegue mais distinguir imagens reais de falsificações.

1.1. Carregar e preparar o conjunto de dados

Você usará o conjunto de dados MNIST para treinar o gerador e o discriminador. O gerador irá gerar dígitos manuscritos semelhantes aos dados MNIST.

1.2. Crie os modelos

Tanto o gerador quanto o discriminador são definidos usando a API Keras Sequential.

1.3. O Gerador

O gerador usa camadas tf.keras.layers.Conv2DTranspose (upsampling) para produzir uma imagem a partir de uma semente (ruído aleatório). Use o gerador (ainda não treinado) para criar uma imagem.

1.4. O discriminador

O discriminador é um classificador de imagens baseado em CNN. Use o discriminador (ainda não treinado) para classificar as imagens geradas como reais ou falsas. O modelo será treinado para gerar valores positivos para imagens reais e valores negativos para imagens falsas.

1.5. Defina a perda e os otimizadores

Defina funções de perda e otimizadores para ambos os modelos.

1.6. Perda do discriminador

Este método quantifica quão bem o discriminador é capaz de distinguir imagens reais de falsificações. Ele compara as previsões do discriminador em imagens reais com uma matriz de 1s e as previsões do discriminador em imagens falsas (geradas) com uma matriz de 0s.

1.7. Perda do gerador

A perda do gerador quantifica quão bem ele foi capaz de enganar o discriminador. Intuitivamente, se o gerador estiver funcionando bem, o discriminador classificará as imagens falsas como reais (ou 1). O discriminador e os otimizadores do gerador são diferentes, pois você treinará duas redes separadamente.

1.8. Definir o loop de treinamento

O loop de treinamento começa com o gerador recebendo uma semente aleatória como entrada. Essa semente é usada para produzir uma imagem. O discriminador é então usado para classificar imagens reais (retiradas do conjunto de treinamento) e imagens falsas (produzidas pelo gerador). A perda é calculada para cada um desses modelos e os gradientes são usados para atualizar o gerador e o discriminador.

1.9. Treine o modelo

Chame o método train() definido acima para treinar o gerador e o discriminador simultaneamente. Observe que treinar GANs pode ser complicado. É importante que o gerador e o discriminador não se sobreponham (por exemplo, que eles treinem em uma taxa semelhante). Restaure o último ponto de verificação.

1.10. Criar um GIF

Use imageio para criar um gif animado usando as imagens salvas durante o treino.

2. Rede Neural Convolucional (CNN)

Este tutorial demonstra o treinamento de uma Rede Neural Convolucional (CNN) simples para classificar imagens CIFAR. Como este tutorial usa a API Keras Sequential , criar e treinar seu modelo levará apenas algumas linhas de código.

Inicie importando TensorFlow, em seguida baixe e prepare o conjunto de dados CIFAR10 e verifique os dados.

2.1. Crie a base convolucional

As 6 linhas de código abaixo definem a base convolucional usando um padrão comum: uma pilha de camadas Conv2D e MaxPooling2D. Como entrada, uma CNN recebe tensores de forma (image_height, image_width, color_channels), ignorando o tamanho do lote. Se você é novo nessas dimensões, color_channels se refere a (R,G,B). Neste exemplo, você configurará sua CNN para processar entradas de forma (32, 32, 3), que é o formato das imagens CIFAR. Você pode fazer isso passando o argumento input_shape para sua primeira camada.

2.2. Exibir a arquitetura do seu modelo até agora:

Acima, você pode ver que a saída de cada camada Conv2D e MaxPooling2D é um tensor 3D de forma (altura, largura, canais). As dimensões de largura e altura tendem a diminuir à medida que você se aprofunda na rede. O número de canais de saída para cada camada Conv2D é controlado pelo primeiro argumento (por exemplo, 32 ou 64). Normalmente, à medida que a largura e a altura diminuem, você pode (computacionalmente) adicionar mais canais de saída em cada camada Conv2D.

2.3. Adicione camadas densas no topo

Para completar o modelo, você alimentará o último tensor de saída da base convolucional (de forma (4, 4, 64)) em uma ou mais camadas Dense para realizar a classificação. Camadas densas recebem vetores como entrada (que são 1D), enquanto a saída atual é um tensor 3D. Primeiro, você achatará (ou desenrolará) a saída 3D para 1D e, em seguida, adicionará uma ou mais camadas Densas no topo. O CIFAR tem 10 classes de saída, então você usa uma camada Dense final com 10 saídas.

2.4. Compilar, treinar e avaliar o modelo

Uma CNN simples alcançou uma precisão de teste de mais de 70%.

API: é a sigla para Application Programming Interface, que é um software intermediário que permite que dois aplicativos se comuniquem. **Exemplo:** Você pode estar familiarizado com o processo de busca de voos online. Assim como o restaurante, você tem uma variedade de opções para escolher, incluindo diferentes cidades, datas de partida e retorno e muito mais. Vamos imaginar que você está reservando seu voo no site de uma companhia aérea. Você escolhe uma cidade e data de partida, uma cidade e data de retorno, classe de cabine, além de outras variáveis. Para reservar seu voo, você interage com o site da companhia aérea para acessar seu banco de dados e ver se algum assento está disponível nessas datas e quais podem ser os custos.

3. Classificação de imagem

Este tutorial mostra como classificar imagens de flores. Ele cria um classificador de imagem usando um modelo tf.keras.Sequential e carrega dados usando tf.keras.utils.image_dataset_from_directory . Neste tutorial é possivel carregar com eficiência um conjunto de dados fora do disco, identificar overfitting e aplicar técnicas para mitigá-lo, incluindo aumento e abandono de dados.

Passo a passo:

1. Examinar e entender os dados;
2. Construir um pipeline de entrada;
3. Construir o modelo;
4. Treinar o modelo;
5. Testar o modelo;
6. Melhorar o modelo e repitir o processo.

3.1. Importar TensorFlow e outras bibliotecas

3.2. Baixe e explore o conjunto de dados

Este tutorial usa um conjunto de dados de cerca de 3.700 fotos de flores. O conjunto de dados contém cinco subdiretórios, um por classe.

3.3. Criar um conjunto de dados

Vamos usar 80% das imagens para treinamento e 20% para validação. Você pode encontrar os nomes das classes no atributo classnames nesses conjuntos de dados. Estes correspondem aos nomes dos diretórios em ordem alfabética.

3.4. Visualize os dados

O image batch é um tensor da forma (32, 180, 180, 3) . Este é um lote de 32 imagens de formato 180x180x3 (a última dimensão refere-se aos canais de cores RGB). O label_batch é um tensor da forma (32,) , estes são os rótulos correspondentes às 32 imagens. Você pode chamar .numpy() nos tensores image_batch e labels_batch para convertê-los em um numpy.ndarray.

3.5. Configurar o conjunto de dados para desempenho

3.5.1. **Dataset.cache** mantém as imagens na memória depois de serem carregadas fora do disco durante a primeira época. Isso garantirá que o conjunto de dados não se torne um argalo ao treinar seu modelo. Se seu conjunto de dados for muito grande para caber na memória, você também poderá usar esse método para criar um cache em disco de alto desempenho.

3.5.2. **Dataset.prefetch** sobrepõe o pré-processamento de dados e a execução do modelo durante o treinamento.

3.6. Padronize os dados

Os valores do canal RGB estão na faixa [0, 255]. Isso não é ideal para uma rede neural; em geral, você deve procurar tornar seus valores de entrada pequenos. Aqui, você padronizará os valores para estarem no intervalo [0, 1] usando tf.keras.layers.Rescaling.

3.7. Crie o modelo

O modelo Sequencial consiste em três blocos de convolução (tf.keras.layers.Conv2D) com uma camada de agrupamento máximo (tf.keras.layers.MaxPooling2D) em cada um deles. Há uma camada totalmente conectada (tf.keras.layers.Dense) com 128 unidades em cima dela que é ativada por uma função de ativação ReLU ('relu').

3.8. Visualize os resultados do treinamento

Os gráficos mostram que a precisão do treinamento e a precisão da validação estão com grandes margens, e o modelo alcançou apenas cerca de 60% de precisão no conjunto de validação.

3.9. Sobreajuste

Nos gráficos acima, a precisão do treinamento está aumentando linearmente ao longo do tempo, enquanto a precisão da validação fica em torno de 60% no processo de treinamento. Além disso, a diferença na precisão entre treinamento e da validação é perceptível - um sinal de overfitting. Quando há um pequeno número de exemplos de treinamento, o modelo às vezes aprende com ruídos ou detalhes indesejados de exemplos de treinamento - a ponto de impactar negativamente o desempenho do modelo em novos exemplos. Esse fenômeno é conhecido como overfitting. Isso significa que o modelo terá dificuldade em generalizar em um novo conjunto de dados. Existem várias maneiras de combater o overfitting no processo de treinamento. Neste tutorial, você usará *o aumento de dados* e adicionará *Dropout* ao seu modelo.

Aumento de dados

Consiste em gerar dados de treinamento adicionais de seus exemplos existentes, aumentando-os, usando transformações aleatórias que produzem imagens. Isso ajuda a expor o modelo a mais aspectos dos dados e a generalizar melhor. Você implementará o aumento de dados usando as seguintes camadas de pré-processamento keras: tf.keras.layers.RandomFlip, tf.keras.layers.RandomRotation e tf.keras.layers.RandomZoom.

Dropout

Consiste em aplicar uma regularização de dropout na rede. Quando você aplica dropout a uma camada, ela descarta aleatoriamente (definindo a ativação como zero) um número de unidades de saída da camada durante o processo de treinamento. Dropout recebe um número fracionário como valor de entrada, na forma de 0.1, 0.2, 0.4, etc. Isso significa descartar 10%, 20% ou 40% das unidades de saída aleatoriamente da camada aplicada.

3.10. Visualize os resultados do treinamento

Depois de aplicar o aumento de dados e tf.keras.layers.Dropout, há menos overfitting do que antes, e a precisão do treinamento e da validação está mais alinhada.

3.11. Prever novos dados

Por fim, vamos usar nosso modelo para classificar uma imagem que não foi incluída nos conjuntos de treinamento ou validação.