UNIVERSIDADE FEDERAL DO PARANÁ
JOÃO FASSINA

GERAÇÃO DE IMAGENS ESTILIZADAS A PARTIR DE TEXTO UTILIZANDO MACHINE LEARNING

CURITIBA
2023

UNIVERSIDADE FEDERAL DO PARANÁ
JOÃO FASSINA

GERAÇÃO DE IMAGENS ESTILIZADAS A PARTIR DE TEXTO UTILIZANDO MACHINE LEARNING

Monografia apresentada como requisito parcial à conclusão do curso de graduação em Matemática Industrial, pela Universidade Federal do Paraná.
Orientador: Prof. Dr. Lucas Garcia Pedroso.

CURITIBA
2023

"Data Scientist (n.): Person who is better at statistics than any software engineer and better at software engineering than any statistician."

Josh Wills

Agradecimentos

Agradeço a minha família que sempre me apoiou em minhas empreitadas, em especial a minha esposa Fernanda Cristina Peixoto Coelho Fassina que acompanhou toda a minha trajetória.

Ao grupo CiDAMO, ao qual eu tive a honra de ser um dos membros fundadores, o que me proporcionou a minha carreira e diversas amizades que vou levar para o resto da vida.

Ao professor Abel Soares Siqueira que sempre me motivou e proporcionou um ambiente de aprendizado além dos limites da sala de aula e ementa.

E por fim, ao meu orientador, o professor Lucas Garcia Pedroso por todo o apoio e carinho não apenas durante a construção deste trabalho mas também desde o começo do curso.

Resumo

Nesta monografia estudamos alguns conceitos de Redes Neurais, como o Perceptron, algumas arquiteturas que revolucionaram o mundo da Inteligência Artificial e o modelo de *Stable Diffusion* com os seus componentes internos a fim de realizar ajustes finos no modelo para chegar em uma versão que gere imagens no estilo de Pokémon. Exploramos também alguns conceitos matemáticos como a convolução, mecanismo de atenção e valores latentes. Pensando no ciclo de vida de um modelo de *Machine Learning*, passamos pelas etapas de anotação de dados, treinamento do modelo e por fim inferência.

Palavras-chaves: *Machine Learning. Stable Diffusion. Pokémon. Processamento de Linguagem Natural. Geração de Imagens. Visão Computacional.*

Abstract

In this monograph we study some concepts of Neural Networks, such as Perceptron, some architectures that revolutionized the world of Artificial Intelligence and the *Stable Diffusion* model with its internal components in order to fine-tune the model to build a version that generates Pokémon-style images. We also explore some mathematical concepts such as convolution, attention mechanism and latent values. Thinking about the life cycle of a *Machine Learning* model, we go through the steps of data annotation, model training and finally inference.

Keywords: *Machine Learning. Stable Diffusion. Pokémon. Natural Language Processing. Image Generation. Computer Vision.*

Sumário

Lista de Figuras

Introdução

Machine Learning é um tema que viu a sua importância crescer exponencialmente nos últimos anos entre modelos preditivos, visão computacional, grandes modelos de linguagem (*ChatGPT*) e geração de imagens a partir de texto. Com foco em resolver problemas reais e aumentar a produtividade onde é possível, já nos vemos cercados por modelos de Inteligência Artificial, seja navegando na internet, fazendo uma compra na padaria ou indo ao médico. Neste texto vamos explorar um pouco do funcionamento do modelo de *Stable Diffusion* e os seus componentes.

O objetivo deste trabalho é a construção de um modelo, usando o *Stable Diffusion* como base, que gere imagens no estilo de Pokémon, permitindo ao longo do caminho também explorar diversos outros conceitos pertinentes à área de *Machine Learning*.

No Capítulo 1 apresentamos alguns conceitos de *Machine Learning* que serão utilizados para a construção do modelo de geração de imagens. No Capítulo 2 são apresentados os componentes que constituem a arquitetura do modelo e como eles interagem entre si. O Capítulo 3 traz informação sobre os treinamentos, experimentos e resultados.

Capítulo 1

Conceitos base

Neste Capítulo apresentaremos o básico de alguns componentes necessários para a construção dos modelos mais à frente, entre eles o simples perceptron e algumas estruturas como redes convolucionais e *ResNet*.

1.1 Perceptron

A unidade mais básica que originou as redes neurais e revolucionou o mundo. Na sua versão original, era nada mais do que um simples classificador binário composto por uma função de decisão que envolvia superar ou não um *threshold* de acordo com a seguinte função:

$$f(x) = \begin{cases} 1, & w \cdot x + b > 0 \\ 0, & \text{caso contrário} \end{cases}$$

onde w é o vetor de pesos da rede, x é o vetor de entrada dos dados e b é o viés do modelo, que não depende das entradas.

O treinamento deste modelo é muito simples e envolve realizar as predições, comparar com o resultado esperado e ajustar os pesos da rede, levando como parâmetro apenas a taxa de aprendizado, que influencia no tamanho do ajuste dos pesos.

Futuramente acabou sendo alterada para um funcionamento mais geral, deixando de olhar apenas para o *threshold* e agora utilizando o que chamamos de funções de ativação que permitem que tenhamos diferentes valores de saída dependendo da necessidade do problema.

Uma das mais famosas e mais utilizadas é a ReLU, que tem uma ideia de ser próxima de um neurônio humano:

$$f(x) = \begin{cases} x, & w \cdot x > 0 \\ 0, & \text{caso contrário.} \end{cases}$$

Além dos resultados ótimos entregues ao utilizar esta função no treinamento, o fato de que ela contém apenas uma comparação também a faz extremamente eficiente perante a recursos computacionais, o que é necessário quando vamos treinar redes profundas.

1.2 Redes Convolucionais

As redes convolucionais são um tipo de redes neurais usadas especialmente em aplicações que envolvem imagens, elas recebem este nome pois utilizam a convolução que olha para a intercessão de duas funções. No nosso caso, como olhamos para matrizes ao invés da intercessão, realizamos uma redução, onde aplicamos uma soma dos elementos de um mapa, multiplicados pelos pesos de nossa rede e transformamos em um único valor.

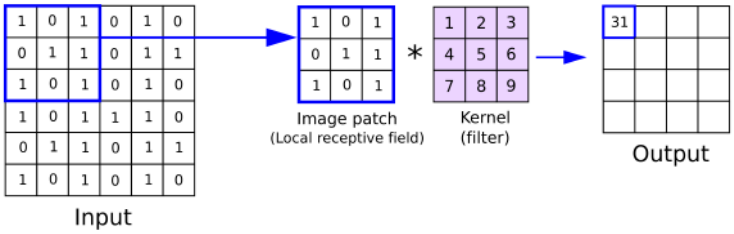


Figura 1.1: Exemplo de uma camada de convolução: blocos sequenciais são selecionados e uma operação é aplicada a eles, tendo como saída uma matriz menor, que concentra a informação da original. Fonte: https://wikidocs.net/164365 Acessado em 18/05/2023.

Uma diferença chave que temos ao utilizar as redes convolucionais é a possibilidade de uso de *padding*, que nada mais é do que o preenchimento de uma camada externa composta por alguma constante (comumente 0), isso permite que seja aplicada a operação de convolução sem reduzir muito, ou sem reduzir nada, o tamanho da nossa matriz.

Outra operação que também foi muito importante para o sucesso das redes convolucionais é o *pooling*, que consiste em uma ideia parecida da convolução, mas desta vez vamos apenas aplicar alguma função no nosso mapa, seja ela pegar o máximo dos dados selecionados ou a média e agora também deixamos de ter sobreposição entre as operações, ou seja um dado só impacta em um resultado.

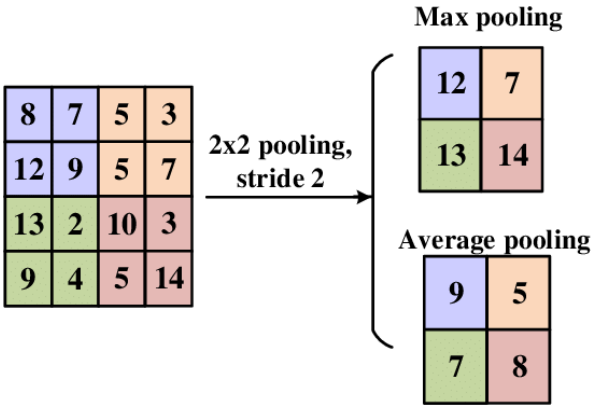


Figura 1.2: Exemplo do funcionamento de dois tipos de *pooling*. No primeiro caso, é selecionado o maior valor dentre a seleção. Já no segundo, é calculada a média dos valores. Fonte: https://www.researchgate.net/figure/Pooling-layer-operation-oproaches-1-Pooling-layers-For-the-function-of-decreasing-the
_fig4_340812216 Acessado em 18/05/2023.

1.3 ResNet

A ResNet é uma arquitetura de redes neurais sugerida em [7], que revolucionou o mundo da Inteligência Artificial. Uma das grandes dificuldades relacionadas a modelos sempre foi a memória de longo prazo, já que a cada camada a informação vai ser alterada de diversos modos e é muito difícil preservar informação ao longo da passagem do dado.

O modo de resolver esse problema foi inserindo o que é chamdo de *skip connections*, nada mais do que literalmente pular algumas camadas da rede e adicionar os valores direto à frente, junto com o resultado das camadas.

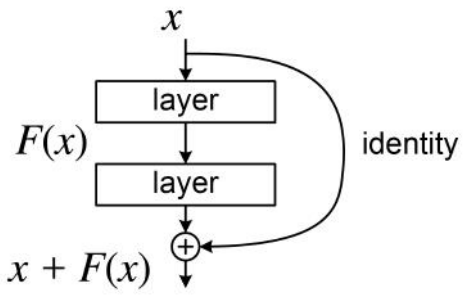


Figura 1.3: Exemplo de *skip connection*. Fonte: By LunarLullaby - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=131458370 Acessado em 18/05/2023.

As *skip-connections* não só foram cruciais em si por permitirem o treinamento de redes profundas com menos esforço e maior acurácia, mas principalmente por originarem algumas das maiores inovações da atualidade no ramo de *Machine Learning*, como os *Transformers* (que foram base para LLM's como o ChatGPT) e o AlphaGo.

1.4 Transformers

Um dos grandes problemas de se trabalhar com informação linguística é que as janelas de contexto necessárias são muito grandes, além de que o sentido de muitas palavras depende completamente de onde ela está inserida. Com isso em mente foi proposto em [8] uma nova arquitetura que combina as redes residuais com informação contextual apurada.

A base para todos os melhores modelos de linguagem recentes são os *Transformers*, desbancando os seus antigos concorrentes, como a LSTM (*Long short-term memory*) [10], melhorando a capacidade de perdurar informações ao longo do treinamento, tendo uma capacidade de paralelização muito superior. O que faz um *transformer* ser tão acurado é o mecanismo de atenção que permite que o modelo foque em certas informações dos dados para previsões específicas.

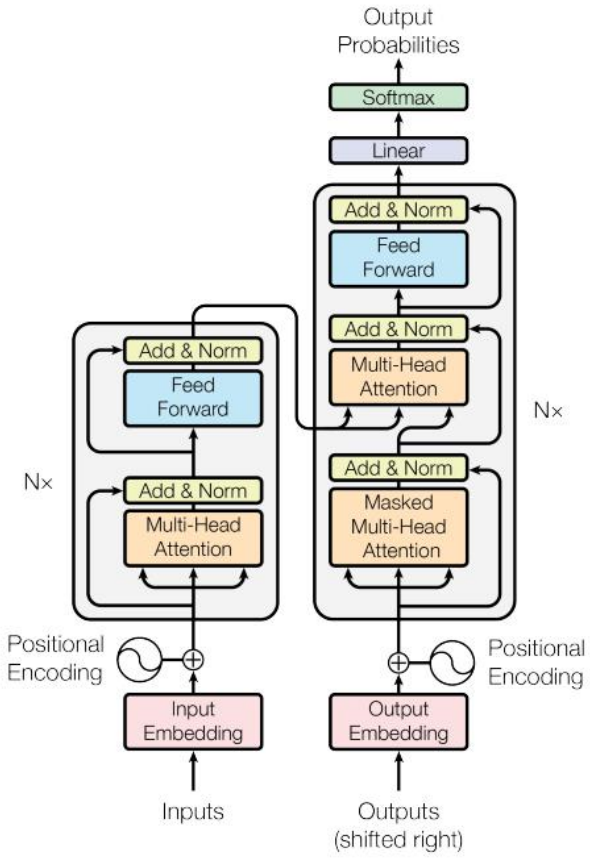


Figura 1.4: Arquitetura do *Transformer*. Existem duas entradas para o *Transformer*, a primeira sendo os *embeddings* de contexto e a segunda sendo a saída do próprio *Transformer*. Fonte: [8] Acessado em 18/05/2023.

O mecanismo de atenção funciona como um vetor de pesos flexíveis, que são alterados ao longo da execução do modelo, fornecemos tanto a entrada, quanto o que já temos de previsão (sempre defasado, assim o modelo não sabe o que ele deve prever nesta instância), realizamos alguns *encodings* posicionais, com a intenção de trazer contexto à

palavra, dada a sua posição na frase e a de outras palavras que a cercam, conseguindo por exemplo diferenciar manga (fruta) de manga (blusa).

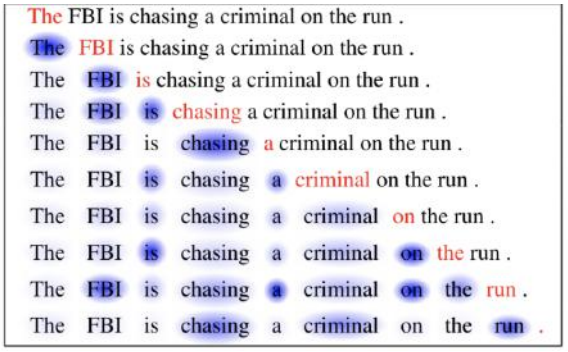


Figura 1.5: Mecanismo de atenção operando na inferência, em vermelho temos a predição e em azul a atenção. Fonte: https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/
Acessado em 18/05/2023.

Deste modo temos ao fim um modelo que consegue propagar parágrafos de informação, consegue definir *embeddings* diferentes para contextos textuais diferentes e que tem a capacidade de selecionar dinamicamente quais serão os valores importantes para a previsão, tudo isso além de ser estável e de treinamento rápido.

1.5 Variáveis latentes

Os valores latentes são observações advindas de variáveis latentes, que podem ser encontradas a partir da relação de outros valores de que temos conhecimento, ou seja, conseguimos utilizar estas variáveis latentes para condensar a informação a partir de um número maior de variáveis.

Podemos ter um caso onde temos 10 variáveis diferentes porém quando fazemos um trabalho de redução de dimensionalidade descobrimos que se utilizarmos 3 variáveis latentes conseguimos ter resultados equivalentes a utilizar as 10, tudo porque os dados carregam relações matemáticas intrínsecas entre si.

Capítulo 2

Stable Diffusion

Neste capítulo vamos explicar um pouco sobre o funcionamento do modelo Stable Diffusion (SD) e as suas estruturas internas, as principais referências deste capítulo são [1], [2], [4], [5],[6] e especificamente sobre a construção do modelo, [3]. Modelos de difusão são cadeias de Markov treinadas com o objetivo de aprender a estrutura latente de um conjunto de dados, ou seja, aprender as relações mais importantes entre as variáveis, deixando de lado alguns detalhes menos importantes. Isso é feito utilizando aplicações sucessivas de ruído gaussiano, construindo uma imagem cada vez mais distante da original. Ao fim dessa etapa iteramos novamente, mas desta vez com o objetivo de retirar o ruído e reconstruir a imagem.

A arquitetura simplificada do modelo se resume a três etapas: VAE (Variational Autoencoders), U-Net e CLIP.

2.1 Variational Autoencoders

O modelo de SD utliza uma variação do modelo de redes neurais de *autoencoders* chamada de *Variational Autoencoder* (VAE) [4], ela compartilha de uma arquitetura muito similiar, quando não igual em muitos casos, da estrutura básica, portanto vamos falar um pouco sobre a versão mais simples antes de falar sobre o VAE.

2.1.1 Autoencoders

Autoencoders são uma estrutura de redes neurais apresentada em [1] com o intuito de aperfeiçoar a redução de dimensionalidade que era feita utilizando PCA (*Principal Component Analysis*), com uma motivação focada especialmente em análise e usabilidade dos dados.

Um grande salto que difere os Autoencoders do PCA é a generalização das funções de mapeamento e reconstrução do espaço latente. Onde antes podíamos utilizar apenas combinações lineares das variáveis conhecidas, agora utilizamos duas funções vetoriais não lineares quaisquer. A primeira reduzindo o tamanho do espaço de variáveis e nos levando ao espaço latente e a segunda reconstruindo o nosso espaço original com uma certa perda.

Do ponto de vista prático essas funções são as camadas da nossa rede.

O nome *autoencoders* vem do modo de treinamento desta rede. Como estamos falando de um modelo de aprendizado supervisionado, precisamos saber exatamente o que estamos inserindo e o que esperamos da saída de nosso modelo. Da primeira função, que faz a redução, sabemos quais são as suas entradas, mas não quais serão as suas saídas, portanto não podemos treinar uma rede apenas para a compressão do espaço. A segunda função é o caso contrário, não sabemos qual é a sua entrada mas sabemos exatamente quais são as saídas esperadas, então o modo encontrado para poder treinar esta estrutura foi a de ligar essas duas redes em seu ponto em comum, o espaço latente, assim conseguimos que a rede aprenda a transformação tendo apenas os dados originais.

Escolhemos qual será o fator de compressão do *encoder* definindo o tamanho da camada de gargalo, que é a saída do *encoder*, assim como no PCA, quão menor o tamanho da camada maior é a perda de informação pelo caminho. No caso do SD as imagens que originalmente seriam matrizes (3, 512 , 512) viram (3, 64, 64) no espaço latente, usando 64 vezes menos memória.

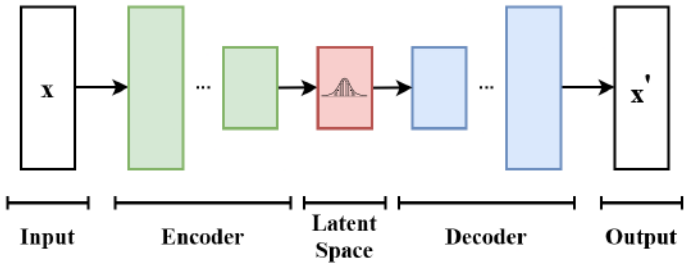


Figura 2.1: Funcionamento de um *autoencoder*. Fonte: EugenioTL - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=107231101 Acessado em 18/05/2023.

2.1.2 Variational Autoencoders

Agora nós deixamos de utilizar apenas funções não lineares quaisquer em nossa rede e entramos no mundo das probabilidades, com os objetivo de evitar *overfitting* e gerar um espaço latente com uma variância menor.

O que antes era codificado ao espaço latente como um simples ponto agora vira uma distribuição dos possíveis valores latentes que poderiam ser gerados partindo do nosso ponto inicial, desta distribuição é feita uma amostra que é decodificada para o espaço original. Por definição essas distribuições são normais, facilitando a nossa avaliação do erro.

Os dois principais motivos para a escolha do uso de um VAE no SD foram a natureza de uma distribuição de probabilidades, que é altamente eficiente para tarefas que

envolvem dados com uma estrutura espacial, e a economia em custo computacional, uma vez que o uso de uma distribuição facilita e agiliza o treinamento do modelo.
Utilizamos como regularização apenas a divergência de Kullback-Leibler (também conhecida como entropia relativa), que mede o quão diferentes duas distribuições são, portanto avaliamos o nosso erro principalmente no espaço latente evitando a utilização de métricas mais complexas em um espaço sem compressão.

Este modelo é crucial para que o SD seja referência dentro do mundo de geração de imagens, pela agilidade com a qual ele viabiliza o treinamento, pela qualidade preservada, já que com o modelo mais rápido não precisamos reduzir tanto as nossas imagens originais, bem como pelo aspecto geracional probabilístico que é inserido no modelo, evitando o uso da estrutura como o de uma GAN, onde apenas escolhemos pontos no espaço latente ao acaso e decodificamos a partir daí.

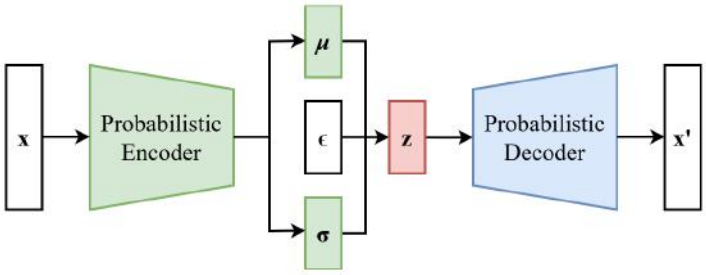


Figura 2.2: Funcionamento de um variational *autoencoder*. A principal diferença está no *encoder* probabilístico, que transforma um valor em uma distribuição de probabilidades do espaço latente. Fonte: EugenioTL - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=107231104 Acessado em 18/05/2023.

2.2 U-net

Redes convolucionais já vinham sendo utilizadas há um tempo antes da criação da U-net, porém com alguns problemas inerentes à sua estrutura e no contexto de sua criação, a segmentação de imagens biomédicas, os principais problemas eram dois: primeiro, uma troca entre acurácia e contexto. Ao utilizar amostras maiores da imagem original precisamos de mais camadas de *max-pooling*, o que acaba reduzindo a acurácia da segmentação. E ao utilizar camadas menores, reduzimos diretamente o nosso contexto. O segundo, a rede era muito lenta. Devido a cada aplicação levar em conta uma amostra do todo, isso nos dá um grande número de iterações por imagem. E além disso essas amostras acabam se intercedendo em vários momentos, fazendo com que o treinamento não seja eficiente.

Com isso em mente, em [5] os autores apresentam algumas propostas para resolver esses problemas, a primeira sendo a suplementação de informações para a rede utilizando uma versão de uma rede de contração inversa, ou seja, uma expansão, onde trocamos as

camadas de *pooling* por camadas de *upsampling*, fazendo com que a saída seja em alta resolução, permitindo que o modelo aprenda de um modo mais efetivo.

Essa rede de *upscaling* é encaixada na saída da rede de contração, formando uma estrutura que lembra o formato de um U. Ao longo desse U, temos também algumas conexões diretas com uma parte mais à frente da rede, fornecendo contexto preservado, sem precisar contar apenas com a rede para fazer este trabalho.

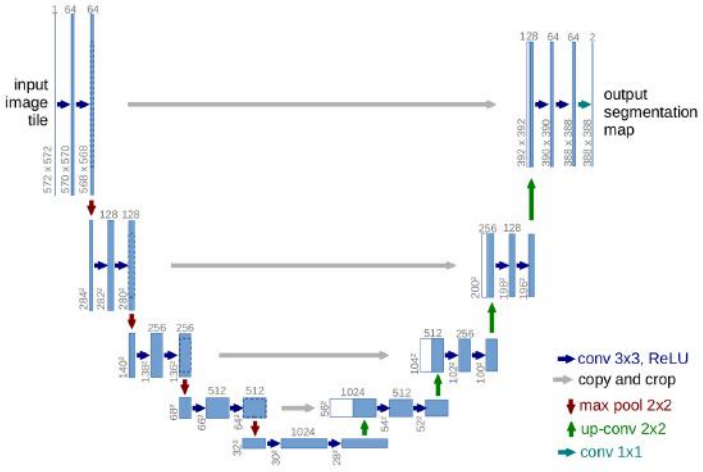


Figura 2.3: Arquitetura do modelo de U-net. Fonte: [5]

A segunda grande alteração foi a de aumentar o tamanho dos *feature channels*, ou seja, aumentar o tamanho do tensor, permitindo que muito mais informação seja propagada ao longo da rede, uma vez que todas as vezes que reduzimos o tamanho da imagem, duplicamos o número de canais.

Uma modificação crucial feita no desenvolvimento do SD foi a utilização de condicionamento para direcionar a atenção da U-net. Foi adicionada uma camada de atenção tanto na compressão quanto na descompressão, que traz informação do nosso *prompt*, ou seja, o nosso objetivo. Conseguimos controlar o quão forte deve ser este direcionamento com um parâmetro do modelo, dando mais liberdade geracional a ele ou restringindo o seu foco a apenas o que foi pedido.

A U-net é utilizada para iterativamente retirar ruído de uma "imagem" inicial, que é apenas ruído gaussiano, como temos o nosso contexto de texto sendo utilizado para condicionar a nossa rede, a cada reconstrução da imagem vamos tendendo a uma imagem final mais coerente com o *prompt* que foi passado. Conseguimos utilizar o número de vezes que a imagem será reprocessada como um parâmetro no SD, nos permitindo ter um certa flexibilidade nesta etapa de remoção de ruído, quando utilizamos um *prompt* mais complexo podemos precisar de mais iterações para que o resultado fique coerente, em *prompts* mais simples, muitas iterações acabam prejudicando a imagem final, pois o modelo passa a deformar uma imagem que já estava boa.

2.3 CLIP

O problema de gerar anotações para imagens já é muito antigo e muito estudado, porém costumavam seguir um mesmo formato de treinar um encoder de imagem para extrair os valores latentes e um classificador linear para predizer a categoria.

Constrastive Language Image Pre-training (CLIP) é o método proposto em [6] que inova perante os anteriores com uma mudança crucial nas estratégias que eram empregadas. Os diversos modelos que já foram estado da arte para o propósito de gerar uma legenda para imagens passaram por diversos fundamentos e motivações distintas, mas todos seguiam um padrão em comum, que era o de predizer palavra a palavra qual era a exata legenda daquela imagem. Esta é uma tarefa muito difícil, já que para algumas imagens muito similares as legendas podem mudar drasticamente de acordo com o anotador. Isso não implica apenas em métricas baixas, o maior problema deste método era o tempo de treinamento dos modelos, como por exemplo a ResNeXt101-32x48d que levou 19 anos de processamento em GPU para ficar pronta, então o CLIP procura encontrar qual o par de frase completa e imagem faz sentido, isso é construído no treinamento tentando otimizar a similaridade cosseno dos pares corretos e minimizar a dos pares incorretos.

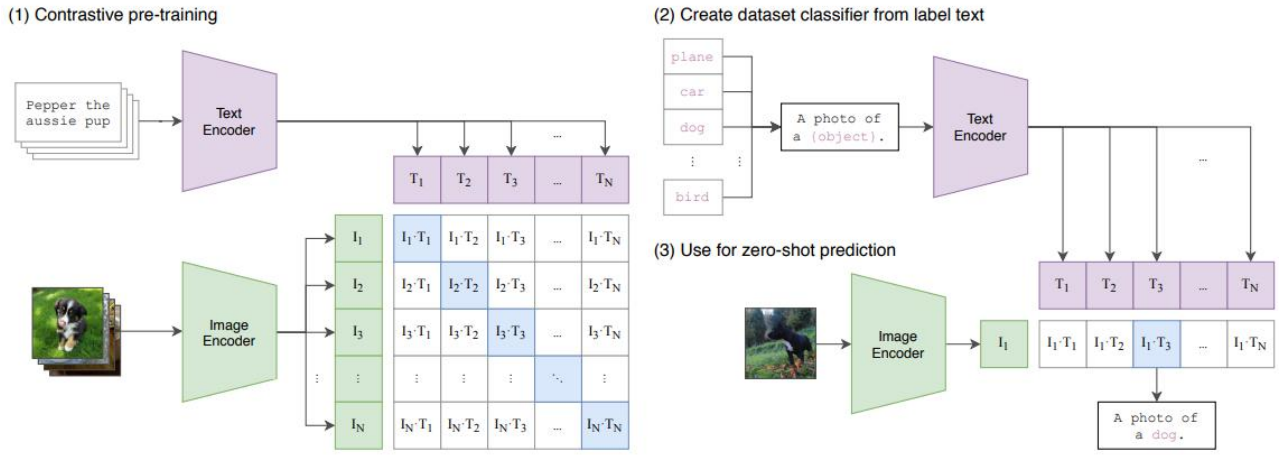


Figura 2.4: Funcionamento do CLIP. Fonte: [6]

A arquitetura desde modelo é bem simples. Ele utiliza um encoder de imagem, que neste caso é uma versão modificada da ResNet, que já se provou extremamente eficiente para este tipo de aplicação e como encoder de texto foi utilizado o Transformer, que segue sendo o estado da arte em diversas tarefas relacionadas a texto.

Um outro benefício de ser um modelo contrastivo é o de podemos utilizar como dataset para treino todo conjunto de imagem e texto (que façam sentido) que for possível conseguir na internet, já que não precisamos nos preocupar com categorias específicas, apenas com pares de imagem e texto quaisquer. O autor chama isso de *Natural Language Supervision*, ou seja, um modelo supervisionado pela linguagem natural.

Desse modo temos uma rede que não está limitada a categorizar imagens em categorias já aprendidas e sim encontrar os pares mais coerentes, permitindo que o modelo seja utilizado em *zero-shot*, ou seja, apenas descobrir quais são as categorias que serão utilizadas na hora da inferência.

Como este modelo é utilizado para gerar a informação de contexto textual para a U-net, é essencial que ele tenha um conhecimento vasto, já que a possibilidade de prompts é essencialmente infinita. Além disso temos também algumas propriedades que o CLIP cumpre muito bem, como a consistência entre frases próximas e noção de sobreposição de elementos ao utilizar com frases que contenham umas as outras. O modelo consegue escolher a versão mais específica caso seja esse o resultado que faça mais sentido.

2.4 Arquitetura

Agora que sabemos como cada um dos nossos componentes funcionam sozinhos, precisamos entender como que a junção dos elementos forma o todo.

Nossa inferência começa com o prompt de texto sendo codificado para os seus valores latentes, ele então é alimentado para a U-net juntamente com a primeira versão da imagem, ruído gaussiano. Com a primeira imagem reconstruída, entramos em um loop de passos do scheduler, que usa como base métodos numéricos para prever a próxima versão da imagem reduzida de ruídos, dada a versão atual da reconstrução condicionada e o resíduo dos ruídos preditos. O número de passos pode ser escolhido como um parâmetro, por padrão são 50, ao fim deste loop teremos a versão final dos valores latentes de nossa imagem, que precisamos apenas alimentar para o nosso decodificador para termos o resultado.

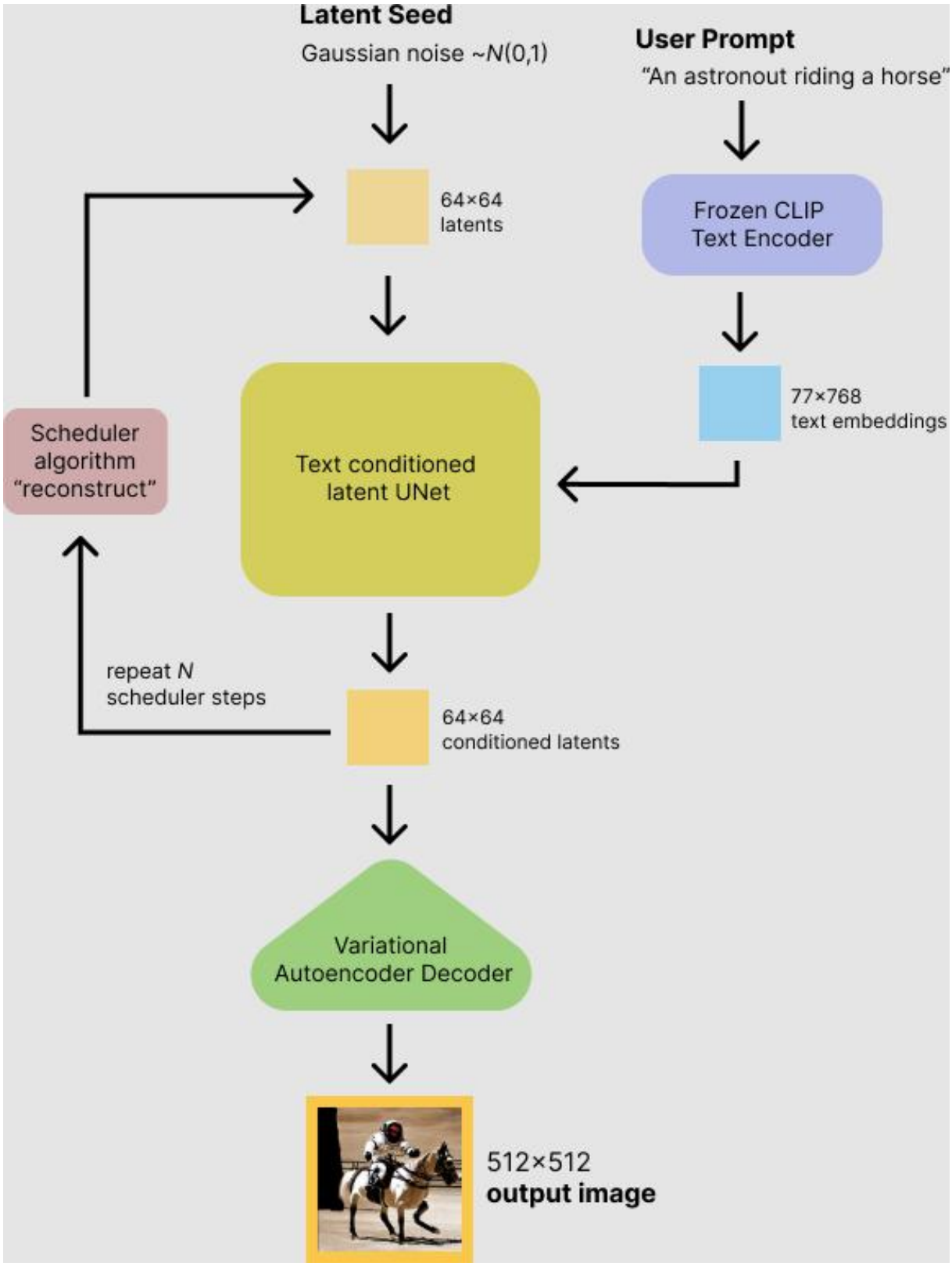


Figura 2.5: Arquitetura simplificada do modelo de SD. Fonte: [3]

Capítulo 3

Experimentos

Neste capítulo vamos falar sobre o treinamento do modelo original, sobre os ajustes finos feitos para conseguirmos imagens com o estilo de Pokémon e comentar um pouco sobre os experimentos realizados e seus resultados.

3.1 Treinamento - Stable Diffusion

Todos os datasets que vamos nos referir nesta seção são disponibilizados pela LAION, uma organização a favor da ciência aberta e são conjuntos do maior dataset disponível, o laion-5B, que contém 5 bilhões de pares (imagem, texto). O treinamento do modelo foi feito utilizando alguns datasets diferentes para algumas etapas distintas:

1. V1.1 - 50% do tempo com o laion-2B-en, que são todas as imagens com texto em inglês disponíveis e 50% com o laion-high-resolution, que são 170 milhões de exemplos de alta resolução.

2. V1.2 - 100% do treinamento retomado do checkpoint anterior com o dataset laion-improved-asthetics, que consiste em um recorte das imagens com maior score artístico, estimado por um outro modelo.

3. V1.4 - Novamente todo o treinamento com um dataset de alto score artístico, mas desta vez um mais refinado.

De acordo com o blogpost do HuggingFace, [9], foram utilizadas 32 clusters de 8 A100, trabalhando em paralelo dentro do ambiente da AWS, sumarizando ao longo do treinamento um total de 150 mil horas de processamento, com um custo estimado de mais de $600.000.

3.2 Treinamento - Nosso modelo

O objetivo do treinamento do nosso modelo era produzir imagens com tema de Pokémon, com um estilo gráfico parecido, pra isso então precisamos de um dataset de imagens de Pokémons com descrição. Existem alguns anotados por modelos porém nenhum com legendas boas, então decidimos anotar as nossas próprias. Foram cerca de 650 imagens que

levaram aproximadamente 40h para serem anotadas. O treinamento do modelo foi feito utilizando as bibliotecas do Hugging Face no ambiente do Lambda Labs, o processamento foi feito com uma A10 24GB e durou 30h.

3.3 Experimentos

Vamos começar comparando resultado obtido utilizando o modelo base para gerar uma imagem e o nosso modelo no mesmo *prompt*.

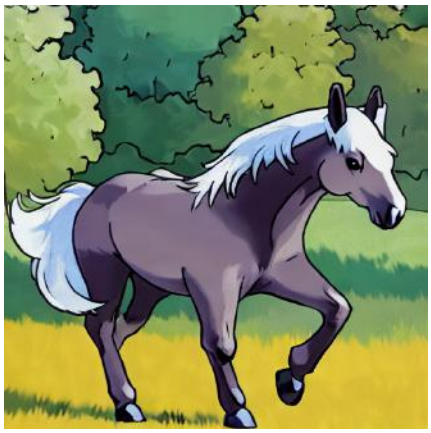
(a) Stable Diffusion.

(b) Nosso modelo.

Figura 3.1: Ambas imagens foram geradas com o *prompt*: "A toucan".

Podemos ver que o nosso treinamento foi um sucesso, o nosso modelo claramente aprendeu a utilizar o estilo de Pokémon para gerar novas imagens. Vamos seguir com mais algumas coisas do cotidiano.

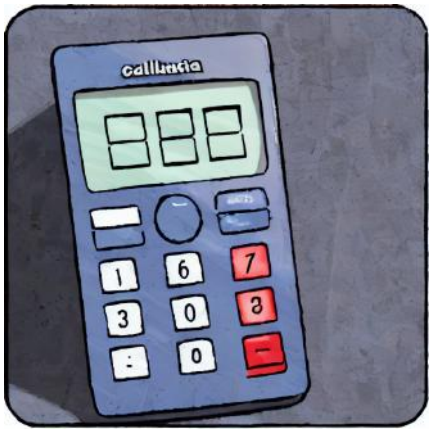
(a) "A capybara".



(b) "A horse".

Figura 3.2: O modelo conhece e performa bem em animais conhecidos.

Até então estamos apenas utilizando *prompts* que não dizem que desejamos uma saída que tenha estilo de Pokémon, então podemos testar a diferença de passar ou não esse pedido.



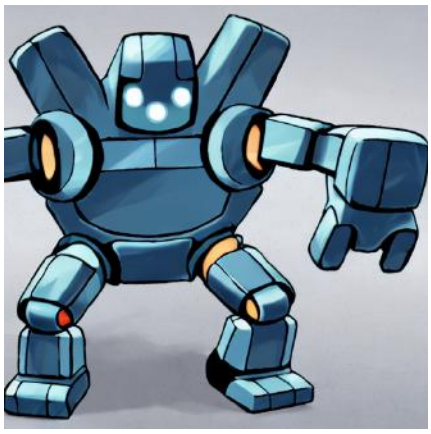
(a) “A calculator”.

(b) “A Pokémon calculator”.

Figura 3.3: Não existe diferença significativa entre as imagens.

Já em alguns outros casos o modelo consegue transformar objetos comuns em Pokémons mais fieis.

(a) “A Pokémon cellphone”.

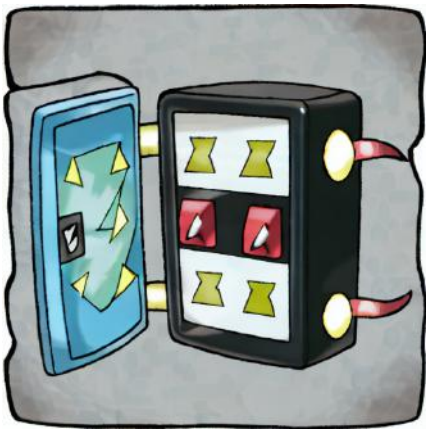


(b) “A humanoid car”.

Figura 3.4: Imagens com maior semelhança a Pokémons.

O modelo conseguiu abstrair o conceito de tipo de um Pokémon e é capaz de juntar isso a objetos que não fazem parte do contexto Pokémon. Conseguimos também misturar pokémons conhecidos pelo modelo.

(a) “A water type fridge”.



(b) “A eletric type fridge”.

(c) “A fire type fridge”.

(d) “A grass type fridge”.

Figura 3.5: Mesmo objeto com tipos diferentes.

(a) “A mix of Pikachu and Rayquaza”.

(b) “A mix of Feraligatr and MewTwo”.

Figura 3.6: Pokémons diferentes unidos.

Como o modelo é probabilistico, um mesmo *prompt* pode gerar imagens diferentes.

(a) “A pokémon band”.

(b) “A pokémon band”.

Figura 3.7: *Prompt* igual, porém com resultados diferentes.

Por fim, algumas das imagens mais fofas que foram geradas.

(a) “A pokémon writing”.

(b) “A rainbow colored Bulbassaur”.

Figura 3.8: Mais alguns exemplos de imagens geradas.

Conclusões

Neste trabalho mostramos o funcionamento de um modelo de *Machine Learning* que transforma texto em imagens, exploramos os seus componentes e como eles funcionam.

Com isso feito foram anotadas manualmente cerca de 650 imagens para poder retreinar as últimas camadas do modelo, com o objetivo de conseguir gerar não apenas imagens a partir de texto, mas imagens no estilo Pokémon.

Chegamos em um modelo que gera imagens estilizadas com qualidade razoável, claramente as limitações de custo e tempo afetaram o resultado do modelo, além de que novos modelos de melhor qualidade estão disponíveis para uso público, possibilitando outro ponto de melhoria para a geração de imagens.

Referências Bibliográficas

[1] Kramer, Mark A. (1991). "Nonlinear principal component analysis using autoassociative neural networks". AIChE Journal.

[2] Fukunaga, K., and W. Koontz, "Application of Karhunen-Loeve Expansion to Feature Selection and Ordering," IEEE Trans. Comput.

[3] Hugging Face .https://huggingface.co/blog/stable_diffusion

[4] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In 2nd International Conference on Learning Representations, ICLR, 2014.

[5] Ronneberger O, Fischer P, Brox T (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". arXiv:1505.04597

[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020

[7] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2015). Deep Residual Learning for Image Recognition. arXiv:1512.03385.

[8] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz; Polosukhin, Illia (2017-06-12). "Attention Is All You Need". arXiv:1706.03762

[9] https://huggingface.co/CompVis/stable-diffusion-v1-4

[10] Hochreiter, Sepp Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.