Classificação de imagens para o desafio “Quick, Draw! Doodle Recognition”

André Moreira Souza
Bruno Mendes da Costa
Felipe Alves Siqueira
Guilherme Amorim Menegali
Guilherme Hideki Tati

Universidade de São Paulo (USP) - Campus São Carlos
Instituto de Ciências Matemáticas e de Computação (ICMC)

Resumo—Quick, Draw! é uma aplicação educativa do Google em forma de jogo, em que deve-se desenhar, em menos de 20 segundos, o tema proposto pela plataforma. A ideia é apresentar a Inteligência Artificial de maneira lúdica.
No jogo a aplicação tenta adivinhar a categoria do desenho. Nosso desafio foi construir um classificador que consiga melhores resultados. Para isso, utilizamos uma Rede Neural Convolucional (CNN) a fim de classificar as imagens. Os resultados obtidos pelo nosso modelo, até então, não atingiram a performance esperada.

Palavras Chave—QuickDraw, Desafio, CNN, Imagens, Classificação, Kaggle.

1 Introdução

Quick, Draw! [1] é um jogo introduzido pela Google em 2016, onde o jogador teria como objetivo desenhar uma figura de um dado objeto enquanto que outro jogador deveria tentar adivinhar qual seria o desenho, assim como no jogo *Pictionary*.

Em 2017 o grupo de pesquisas do Google, Magenta, teve a ideia de dar um passo adiante e a partir dos dados rotulados do *Quick, Draw!* construiu-se uma base para treinamento do modelo *Sketch-RNN*. Desse modo seria possível predizer o desenho ao mesmo tempo em que ele fosse sendo desenhado, logo não seria mais necessário dois jogadores para jogar.

O jogo, atualmente, consiste no usuário usar o mouse, ou os próprios dedos (no caso de *smartphones* ou *tablets*) para desenhar uma série de 6 desenhos, com um tempo limite de 20 segundos para cada desenho, o objetivo do modelo do Google é de tentar adivinhar o desenho feito pelo jogador nesse dado tempo limite. O jogo hoje em dia já possui mais de 1 bilhão de desenhos (ou como eles apelidaram: *Doodles*)

O *Sketch-RNN* [2] é um modelo de Rede Neural Recorrente (RNN) que consegue construir desenhos de objetos comuns baseado em traços. A rede neural foi treinada a desenhar a partir dos milhões de dados coletados do *Quick, Draw!*. O usuário começa a desenhar o objeto e a rede neural o completa.

Tanto o *Sketch-RNN* quando o *Quick, Draw!* são experimentos do "AI Experiments" [3] do Google, um mostruário de simples experimentos em *machine learning*, que misturam imagens, desenhos, músicas, *etc* de modo que todos possam ter contatos com a área de forma amigável.

1.1 Motivação

Ao nos reunirmos para decidir qual seria o tema e o problema a ser abordado neste projeto ainda não tínhamos uma ideia muito clara de sua concepção. Assim, optamos por realizar uma pesquisa no Kaggle [4], plataforma de

Manuscrito entregue em 13/11, 2018.

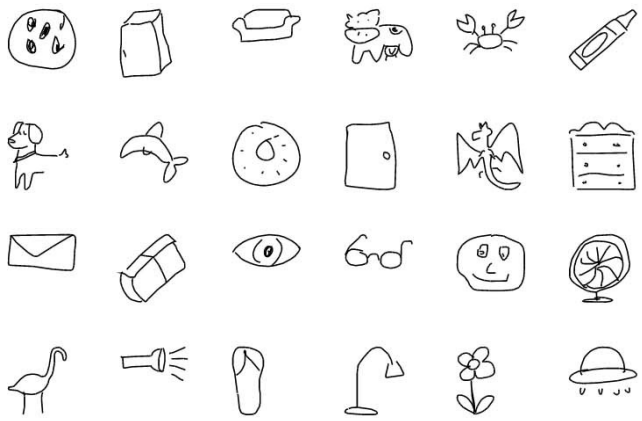

Figura 1. Ilustração dos *Doodles* coletados pela ferramenta.

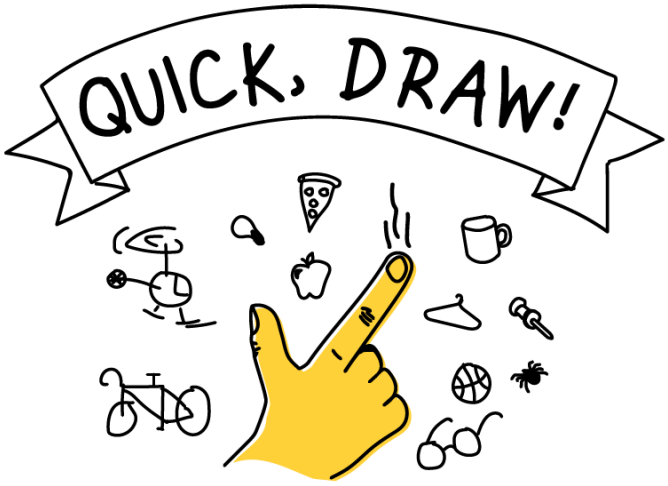

Figura 2. Logotipo do “Quick, Draw!”

competições amplamente conhecida na área de Ciência de Dados.

No Kaggle vimos uma oportunidade tanto para realizar o trabalho da disciplina de Inteligência Artificial, como também relacionar o conteúdo aprendido em sala de aula com os problemas do mundo real. Dito isto, filtramos nossa busca por uma competição que ainda estivesse ativa na plataforma e que ficasse aberta até pelo menos o mês de Dezembro, assim também poderíamos agregar esse desafio em nosso currículo. Entre outros requisitos, das opções restantes acabamos por encontrar uma competição pública patrocinada pelo Google, o *Quick, Draw! Doodle Recognition Challenge* [5].

A competição *Quick, Draw! Doodle Recognition Challenge* iniciou em 26 de Setembro de 2018, patrocinada pelo Google, oferecendo um prêmio de US$12.000 para o primeiro lugar, US$8.000 para o segundo e US$5.000 para o terceiro, com número máximo de 8 membros por time e término em 04 de Dezembro de 2018.

1.2 “O quão preciso você consegue identificar um doodle?”

O desafio proposto na competição se baseia no fato de que como os dados de treinamento vêm do próprio jogo, os desenhos podem estar incompletos ou podem não corresponder ao rótulo. A tarefa então seria construir um classificador melhor para o conjunto de dados do Quick, Draw! que possa efetivamente aprender com os dados ruidosos e mesmo assim obter bons resultados.

Tabela 1: Estrutura do *dataset* simplificado para treino

Campo	Tipo	Descrição
key_id	*uint64*	Identificador único por desenho
countrycode	*string*	Código do país do jogador
drawing	*string*	Desenho vetorizado
timestamp	*datetime*	Data da criação do desenho
word	*string*	Categoria solicitada ao jogador

2 Descrição do Conjunto de Dados

Os dados que utilizamos foram disponibilizados pelo Kaggle para a competição *Quick, Draw! Doodle Recognition Challenge* [5]. O conjunto original foi oferecido pela Google, e coletado a partir de dados do jogo *Quick, Draw!* [6]. No total, o conjunto é composto por aproximadamente 50 milhões de imagens, divididas em 340 classes.

Na competição, foram dados os conjuntos *raw*, com todos os dados vetorizados não processados e totalizando 65,87GB, e *simplified*, onde foi aplicado o algoritmo *Ramer-Douglas-Peucker*, removendo pontos redundantes da imagem e subsequentemente reduzindo o tamanho do *dataset* para 7,37GB. Enfatizamos que ambos os conjuntos fornecem efetivamente a mesma informação.

Um valor do campo *drawing* do conjunto de dados, com n arestas e m pontos em uma aresta, tem a forma $(s_1, s_2, ..., s_n)$, onde s_i, $i = 1, 2, ..., n$, são as arestas do desenho, com formato $((x_1, x_2, ..., x_m), (y_1, y_2, ..., y_m), (t_1, t_2, ..., t_m))$, representando as coordenadas e instante de tempo t_j, $j = 1, 2, ..., m$, em que cada ponto (x_j, y_j) foi percorrido pelo usuário.

2.1 Conjuntos de treinamento e validação

Devido a limitações de *hardware*, especialmente a carência de memória *RAM*, decidimos por utilizar o conjunto "simplificado", representado na tabela 1. Além disso, devido às mesmas limitações, foi necessária a redução do conjunto utilizado de 49.707.919 para 1.553.120 amostras (cerca de 3,1% do conjunto original), contendo 4.568 amostras por classe.

Sobre os dados selecionados, aplicamos a técnica de *Holdout* para validar os resultados do nosso processo, ou seja, dividimos os dados selecionados em 80% para treinamento e 20% para validação.

2.2 Conjunto de teste

O conjunto utilizado para testes foi fornecido pelo Kaggle, para a eventual submissão das predições do nosso modelo na plataforma, contendo aproximadamente 112 mil desenhos vetorizados para classificação. O atributo classe deste conjunto não foi disponibilizado ao público.

2.3 Pré-processamento

Para utilizar a abordagem desejada, houve a necessidade de rasterizar as imagens, ou seja, convertê-las do formato vetorial para o formato *raster* (imagem formada por *pixels*). Com este objetivo, utilizamos uma função, disponibilizada publicamente no Kaggle que toma como entrada o conjunto de arestas de um desenho, e realiza este processo [7]. Nesta função, foram desconsiderados os elementos t_i das arestas e, ao final do processo, normalizamos os valores para o intervalo [0, 1], de maneira que um *pixel* tem valor 1 se faz parte do desenho, e valor 0 caso contrário. Por limitações de *hardware*, ao rasterizar, definimos a dimensão das imagens geradas para 32x32 *pixels*.

Devemos enfatizar que, devido à alta carga de processamento gerada pela rasterização, esta se tornou um gargalo no nosso processo, e foram necessárias alterações para rasterizar os dados em lotes, e minimizar o custo de memória.

3 Abordagem Utilizada

Para resolver o problema foi optado pelo uso de uma Rede Neural Convolucional (CNN) pois possuem três propriedades importantes que permitem um bom desempenho com dados do tipo imagem.

- **Compartilhamento de pesos entre neurônios de um mesmo filtro:** o uso de uma rede totalmente conectada é tipicamente inviável quando o conjunto de dados possuem imagens, pois um pequeno aumento de resolução gera uma explosão na quantidade de parâmetros ajustáveis no modelo, tornando o custo computacional necessário para o treinamento impraticável. Uma CNN divide os mesmos pesos entre neurônios de um mesmo filtro, reduzindo muito a quantidade de parâmetros ajustáveis do modelo. Estes filtros também são chamados de *Kernel*.
- **Campo receptivo de cada neurônio delimitado:** os nós de uma camada de convolução de uma CNN possuem acesso a uma região limitada aos valores do volume de entrada da camada, obrigando-os a levarem em consideração só e somente esta região para o cálculo de seu gradiente para se ajustar na etapa *backward* do algoritmo de descida de gradiente *backpropagation*, utilizado no treinamento do modelo. Esta propriedade, além de reduzir o número de conexões no modelo, produz um efeito de correlação entre *pixels* próximos e dissimilaridade entre *pixels* distantes, propriedades que imagens geralmente possuem.
- **Invariância espacial:** todos os filtros treinados são aplicados em todas as regiões de interesse da imagem, salvo quando há conhecimento externo do domínio de dados

que permite a poda de determinadas regiões a fim de simplificar o modelo como, por exemplo, conhecer que um determinado padrão só pode ocorrer nas bordas da imagem e nunca no centro da mesma. Esta propriedade permite que a CNN reconheça um padrão aprendido na imagem independentemente de sua posição. Em outras palavras, padrões podem sofrer qualquer translação sem afetar a performance do modelo preditivo.

3.1 Arquitetura da CNN

Uma arquitetura de uma CNN convencional possui tipicamente dois grandes "módulos": "extração de características" e "predição".

3.1.1 Etapa de extração de características

O primeiro módulo objetiva só e somente extrair padrões latentes no domínio do conjunto de dados de treinamento, formado tipicamente por camadas de Convolução, *Pooling* e Funções de Ativação. Este primeiro módulo é necessário pois, como mencionado anteriormente, o uso de uma rede totalmente conectada é impraticável devido ao alto custo computacional agregado. Sendo assim, uma maneira de levar em consideração todo o volume de entrada é treinar filtros que reconhecem padrões geralmente não previamente conhecidos no domínio de entrada, e então utilizar somente estas características em camadas totalmente conectadas, que por sua vez terão muito menos parâmetros ajustáveis, para efetivamente realizarem a separação no domínio de dados. Estes filtros são treinados pelo algoritmo de otimização *backpropagation*, que faz uso da descida do gradiente para calcular as taxas de ajuste de cada posição de cada filtro.

3.1.2 Etapa de predição

A segunda etapa, "predição", utiliza de camadas densamente conectadas para separar com hiperplanos o domínio do conjunto de dados de entrada e efetivamente classificar cada uma das instâncias. É importante ressaltar que, sendo tipicamente uma das últimas camadas da arquitetura de uma CNN, a entrada desta camada será algumas das características selecionadas do domínio original de entrada pelas camadas da etapa anterior, permitindo que a quantidade de parâmetros ajustáveis, que estão concentrados neste tipo de camada, seja muito menor e, portanto, o custo computacional necessário para ajustá-los seja tangível.

A arquitetura utilizada possui 12 camadas, agregadas de modo sequencial, como pode ser visto na figura 3. As primeiras camadas são, nesta ordem, convolução, ativação ReLU e *Max Pooling*. Esta mesma sequência se repete uma vez, sendo a única mudança à quantidade de filtros da segunda camada de convolução, que efetivamente dobra, sendo a primeira contendo 32 filtros e a segunda 64 filtros. Em seguida, há uma camada de achatamento, de modo a remover o volume da última camada de *Pooling* para uma única dimensão, de modo que seja possível alimentar a próxima camada, que é totalmente conectada (camada densa) com 512 neurônios. Por fim, há uma camada de saída, onde é aplicado a função *Softmax* (equação 3).

3.2 Camadas de Convolução

Um dos objetivos centrais de uma CNN é aprender automaticamente filtros que podem ser aplicados sobre os dados a fim de extrair características latentes do mesmo. A camada de convolução é responsável por aplicar estes filtros no volume de entrada e gerar um volume de saída contendo o resultado das operações de convolução em relação a toda a entrada. Nossa arquitetura projetada possui duas camadas de convolução, onde o tamanho dos filtros optados foram 4x4 com deslocamento (*stride*) de uma unidade para ambas dimensões (horizontal e vertical). A primeira camada apresenta 32 filtros e a segunda, 64, a fim de aproveitar o volume decrescente de entrada entre as camadas por conta da camada de *Pooling* entre elas.

A implementação foi efetivada utilizando a *Application Programming Interface* (API) Keras, utilizando a linguagem Python.

3.3 Camadas de ativação

Após cada camada de convolução, há uma camada de ativação utilizando a função *Rectified Linear Unit* (ReLU) (Equação 1) com a

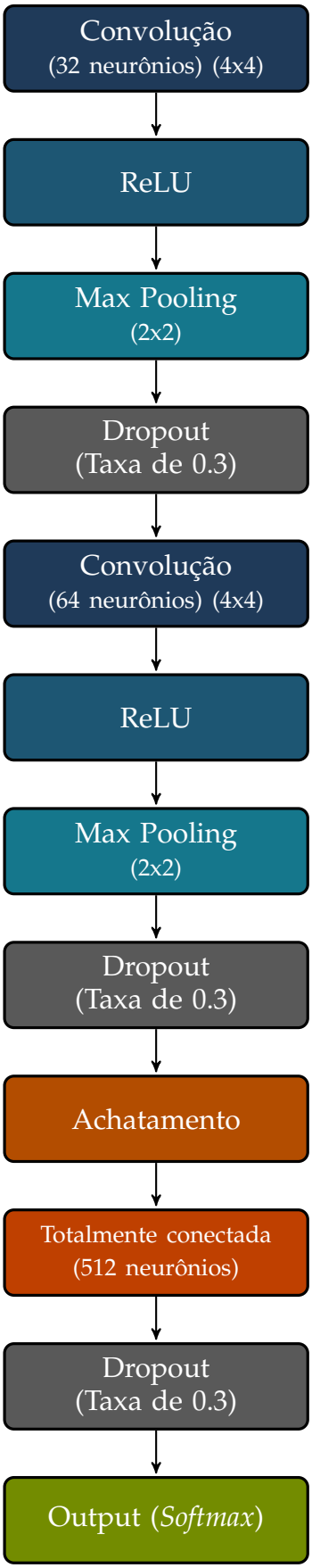

Figura 3. Arquitetura projetada para classificação das imagens.

finalidade de inserir a não-linearidade nos resultados obtidos nas operações de convolução, dado que o domínio de dados de imagem é altamente não-linear. A função ReLU é preferível pois tipicamente agrega custo computacional menor na etapa de treinamento em relação a outras funções de ativação comumente utilizadas, como a Sigmoide ou Tangente Hiperbólica. Além disso, a função ReLU não apresenta o problema de saturação (gradiente zero para valores muito positivos ou negativos) que ambas as outras funções citadas sofrem. Entretanto, a ReLU possui o problema conhecido como "Dying ReLU", onde neurônios podem permanentemente serem desativados por conta do gradiente nulo para valores negativos; Uma das soluções seria o uso da "Leaky ReLU" (Equação 2), onde a parte negativa da função tem um coeficiente angular α não nulo muito pequeno, sendo suficiente para evitar a desativação permanente dos neurônios. Entretanto, nossos testes não indicaram mudanças expressivas na performance do modelo e, portanto, foi mantida a função de ativação ReLU.

$$ReLU(x) \;=\; max(0,\; x) \tag{1}$$

$$LeakyReLU(x) = \begin{cases} x & \text{se x} > 0 \\ \alpha x & \text{c.c.} \end{cases} \tag{2}$$

3.4 Camadas de *Max Pooling*

As camadas de *Max Pooling* são usadas como uma forma de seleção de atributos, onde efetivamente apenas as características com maior correlação com o volume de entrada são escolhidas para formar o volume de saída da camada. Como consequência, o volume de saída torna-se tipicamente muito menor que o volume de entrada e, portanto, o custo computacional de treinamento torna-se menor, permitindo o modelo focar-se apenas nas características mais pertinentes nas próximas camadas. Os filtros da camada de *Max Pooling* são 2x2, com deslocamento (*stride*) de 2 unidades em ambas as dimensões (horizontal e vertical).

3.5 *Dropout*

A técnica *Dropout* é uma técnica de regularização, aplicada para prevenir que o modelo treinado atinja o estado de *overfitting*, isto é, faça um superajustamento em função do conjunto de dados, representando uma função muito mais complexa que o necessário para descrever as partes fundamentais do domínio de entrada e, portanto, não generalizando bem para dados nunca vistos. Esta técnica essencialmente desativa neurônios de forma aleatória com uma probabilidade p, de modo que, essencialmente, 2^n redes diferentes são treinadas simultaneamente, sendo n o número de neurônios afetados pelo *dropout*. Durante a fase de testes, as saídas de todos os neurônios são tipicamente ponderadas pelo fator p (utilizado para desativar os neurônios) para que seja necessário testar apenas a ponderação da rede. O desativamento de alguns neurônios força a rede a não depender de apenas alguns neurônios que podem a vir conseguir uma vantagem de relevância nas primeiras iterações durante a fase de treinamento. As camadas de *dropout* foram utilizadas após as camadas de *pooling* e da camada densamente conectada escondida, com uma taxa fixa de $p = 0.3$.

3.6 Camada densa ("Fully connected")

A camada densamente conectada é responsável por efetivamente separar as instâncias de cada classe utilizando hiperplanos. Os parâmetros ajustáveis deste tipo de camada são necessariamente os coeficientes que descrevem cada hiperplano. Dois pontos devem ser ressaltados nesta etapa: a separação das instâncias não é feita baseada nas imagens rasterizadas (entrada original da rede convolucional), mas sim a partir das características extraídas destas pelas camadas anteriores. Depois, é importante lembrar que a entrada de camadas densamente conectadas são unidimensionais, sendo necessário uma camada intermediária de Achatamento (*Flatten*) entre a última camada de extração de características (que possui volume) e a primeira camada totalmente conectada. Foram utilizados 512 neurônios nesta camada.

key_id	drawing
90003627287624	hand
90052667981386	parrot
90087586309806	owl
90096661653918	shark
90092580281382	parrot

Tabela 2: Subconjunto de instâncias representadas pelas suas respectivas classes através da coluna "drawing".

key_id	parrot	shark	(+336)	owl	hand
90003627287624	0	0	...	0	1
90052667981386	1	0	...	0	0
90087586309806	0	0	...	1	0
90096661653918	0	1	...	0	0
90092580281382	1	0	...	0	0

Tabela 3: Tabela meramente ilustrativa que exemplifica a decodificação das classes de cada instância do conjunto de treinamento, apresentando o mesmo subconjunto da tabela 2 com a coluna "drawing" codificada ("One-hot Encoding").

3.7 Camada de saída

A última camada do modelo é a camada de saída, sendo a quantidade de neurônios correspondendo exatamente ao número de classes do conjunto de dados (340). Cada neurônio de saída é especializado em detectar uma única classe possível do conjunto de dados, característica presente devido a decodificação dos rótulos das instâncias para 340 colunas binárias ("*dummie variables*") que indicam, para cada instância, 1 se ela pertence àquela classe e 0 caso contrário (ver exemplo apresentado com as tabelas 2 e 3). Toda instância do conjunto de dados pertence a exatamente uma única classe.

As saídas de todos os neurônios de saída são então normalizadas pela função *Softmax* (Equação 3), para normalizar as probabilidades de cada imagem pertencer a cada classe, de modo que passem a somar exatamente 1.0.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_k e^{x_k}} \tag{3}$$

3.8 Etapa de treinamento

A etapa de treinamento foi conduzida com o otimizador Adam [8], técnica de descida de gradiente com passo adaptativo, que leva em consideração o primeiro e segundo momentos

(média e variância, respectivamente). Esta escolha foi dada pura e simplesmente com base em testes de outras pesquisas que comparavam este método de ajuste de pesos com outros, como o *RMSProp*, *AdaDelta* ou o *Stochastic Gradient Descent* com otimizador Nesterov, com treinamento em *batchs* de 4096 para reduzir o custo computacional envolvido nesta etapa.

Dos 50 milhões de instâncias do conjunto fornecido pelo Kaggle, apenas ~3% foi utilizado para esta etapa, sendo particionado através da técnica *Holdout* com 80% desta quantidade dedicada ao treinamento do modelo e 20% dedicados a um conjunto de validação.

A função de perda utilizada foi a "*categorical cross-entropy*", cujos valores estão no intervalo [0, 1], sendo interpretados como probabilidades.

Foram utilizadas 40 épocas de treinamento. Foi também utilizada a *Early Stopping*, outra técnica de regularização para reduzir as chances do modelo sofrer de *Overfitting* (além do *Dropout*). A "paciência" utilizada, isto é, o número de iterações consecutivas cuja acurácia do conjunto de validação pode piorar antes da interrupção da etapa de treinamento foi de 15% do número total de épocas, totalizando 6 iterações.

Não foram utilizadas as "meta-informações" relacionadas a cada instância (país de origem, *timestamp*, etc) pois isso demandaria um modelo mais complexo que uma CNN tradicional. Este pode ser um candidato a ponto de melhoria em uma solução mais sofisticada.

4 Resultados

A métrica de validação utilizada foi a "acurácia top 3", isto é, conta-se como um acerto caso a classe de uma instância de teste está entre as três classes mais prováveis na saída gerada pelo modelo preditivo. Não há distinção entre estas três posições de ranking, isto é, acertar a classe da instância com o palpite mais provável é tão válido quanto acertar esta classe com o terceiro palpite mais provável. Este método de validação foi dado pela própria competição. Calculando a "acurácia top 3" com o conjunto de validação, obtivemos no máximo 70,41%. Entretanto, ao submeter no Kaggle obtivemos uma "acurácia top 3" menor, de 59,70%. Vale enfatizar que a submissão ao Kaggle utiliza um conjunto de testes particular fornecido pela própria competição, e não há acesso das classes verdadeiras de cada instância deste conjunto, sendo impossível utilizá-lo como conjunto de validação durante o treinamento do modelo.

Consideramos que os resultados não foram satisfatórios, porque os resultados foram inferiores a centenas de outras submissões na competição. Entretanto, isto já era esperado, pois ainda estamos nos desenvolvendo no aprendizado desta área.

5 Limitações e possíveis melhorias

Durante o treinamento do modelo tivemos certas complicações, como a limitação de memória RAM durante o processo de rasterização das imagens, que ocorreu devido ao grande volume de imagens disponíveis. Isto afetou diretamente os resultados, já que por este motivo não foi possível treinar com todo o conjunto de dados. Se tivéssemos acesso a recursos melhores (RAM e GPU), esta situação seria atenuada e possivelmente poderíamos obter resultados mais satisfatórios, já que acabamos utilizando apenas ~3% dos dados fornecidos.

Além disso, temos as informações que não foram consideradas neste trabalho, como em quais instantes os usuários desenharam cada traço e dados relacionados ao usuário em si, como o país de origem. Uma alternativa seria trocar a nossa camada completamente conectada por uma Rede Neural Recorrente (RNN), assim estas informações dos instantes dos traçados poderiam ser consideradas, já que a RNN possui memória, e portanto tem a noção de tempo. Isto talvez possibilitasse melhorias nos resultados.

6 Conclusão

Este trabalho possibilitou que pudéssemos ter contato com temas atualmente muito relevantes da área de Inteligência Artificial como Deep Learning, Redes Neurais, CNN e Reconhecimento de Imagens.

Além disso, conseguimos entender todos os conceitos por trás de uma Rede Neural Convolucional.

Temos conhecimento de que a CNN tem o potencial de gerar resultados melhores do que os obtidos, mas este primeiro contato foi um importante passo para o nosso aprendizado.

AGRADECIMENTOS

Agradecimentos especiais aos contribuintes do trabalho, ao Cluster da Google, com apoio de hardware para o treinamento dos dados, ao Instituto de Ciências Matemáticas e de Computação (ICMC) e ao Massachusetts Institute of Technology (MIT) com apoio ao ensino através de aulas gratuitas disponibilizadas no *YouTube* e na plataforma online *MIT OpenCourseWare* [9] e ao Kaggle pelo suporte à competição.

REFERÊNCIAS

[1] "Quick, draw!" [Online]. Available: https://quickdraw.withgoogle.com/
[2] "Sketch-rnn." [Online]. Available: https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html
[3] "Ai experiments." [Online]. Available: https://experiments.withgoogle.com/collection/ai
[4] "Kaggle." [Online]. Available: https://www.kaggle.com
[5] "Quick, draw! doodle recognition challenge | kaggle." [Online]. Available: https://www.kaggle.com/c/quickdraw-doodle-recognition/
[6] "The quick, draw! dataset." [Online]. Available: https://github.com/googlecreativelab/quickdraw-dataset
[7] "Image-based cnn | kaggle." [Online]. Available: https://www.kaggle.com/jpmiller/image-based-cnn
[8] "Gentle introduction to the adam optimization algorithm for deep learning." [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
[9] "Mit opencourseware." [Online]. Available: https://ocw.mit.edu/index.htm