Uso de Algoritmos Genéticos para Otimizar os Hiperparâmetros de uma Rede Neural Convolucional

Edson Fagner da Silva Cristovam[1]

[1]Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)
Rua Dom Manoel de Medeiros, s/n – Campus Dois Irmãos – PE – Brasil

fagner.cristovam@hotmail.com

Resumo. *Este artigo tem como objetivo relatar o processo de otimização dos hiperparâmetros de uma rede neural convolucional usando algoritmos evolucionários, mais precisamente, algoritmos genéticos. A rede em questão será treinada com o objetivo de classificar gestos ASL a partir de imagens de mãos.*

1. Introdução

O uso das Rede Neurais Artificiais (*Artificial Neural Network — ANN*), comumente chamadas de Redes Neurais (*Neural Networks*), vem crescendo desde a criação do Perceptron, representação matemática de um neurônio biológico, por Frank Rosenblatt em 1957, descrito no trabalho "The Perceptron — a perceiving and recognizing automaton" [Rosenblatt 1957]. Partindo dessa base, o Perceptron — que resolve problemas linearmente divisíveis —, e com o aumento do poder computacional dos processadores, muitos outros tipos de redes neurais foram desenvolvidos, aumentando em tamanho (número de neurônios, que podem ser distribuídos paralelamente, formando uma camada, que por sua vez podem ser conectadas em série — múltiplas camadas) e, consequentemente, crescendo em complexidade.

A Rede Neural Convolucional (*Convolutional Neural Networks - CNN*) é um tipo de *ANN* bastante usanda no reconhecimento de imagens devido ao seu desempenho. Ela faz parte do subcampo do Aprendizado de Máquina conhecido como Aprendizado Profundo (*Deep Learning*) — que é um especialista com um conjunto extremamente complexo usado para obter resultados melhores com o mesmo conjunto de dados. A *CNN* é um ótimo exemplo de variação do uso para o Perceptron multicamadas empregado em solucionar problemas de classificação. Esse tipo de rede tem topologia de grade para processamento de dados, onde os pontos de dados são processados em uma correlação espacial entre os pontos de dados vizinhos. Ainda nesse sentido, a rede neural convolucional usa o método de convolução em oposição à multiplicação geral da matriz em pelo menos uma das camadas [Sharma 2020].

Antes de iniciar o treinamento de uma rede é necessário definir alguns atributos que influenciam a qualidade e velocidade do modelo e determinam sua topologia. Esses parâmetros são conhecidos como hiperparâmetros (*Hyperparameters*). Exemplos de hiperparâmetros são: taxa de aprendizagem, número de épocas e *momentum* [DeepAI 2020]. Dessa forma, a quantidade de parâmetros aumenta conforme mais atributos são necessários para representar a rede.

Encontrar os melhores hiperparâmetros é uma tarefa de otimização. Otimização se refere a descobrir uma ou mais soluções possíveis que correspondem aos valores extremos de um ou mais objetivos. Quando o problema de otimização envolve explorar mais

que uma função-objetivo, a tarefa de encontrar uma ou mais soluções ótimas é conhecida como otimização multiobjetivo [Deb 2001]. Algoritmos Evolucionários (*Evolutionary Algorithms — EA*) são eficientes para otimizar valores, especialmente sua extensão, os Algoritmos Genéticos (Genetic Algorithm — *GA*). Uma aplicação real de *EA* para encontrar múltiplos valores e que obteve êxito foi sugerida por David Schaffer em 1984 [Deb 2001].

1.1. Objetivo

O objetivo deste trabalho é usar algoritmos genéticos para otimizar os hiperparâmetros de redes neurais convolucionais. Essas redes serão treinadas para identificar imagens de gestos ASL gerados pela Massey University [Barczak et al. 2011].

2. Metodologia

Para o treinamento das redes neurais convolucionais foi usado um conjunto de dados com imagens de mãos fazendo gestos ASL em um fundo preto. Essas imagens foram disponibilizadas pela Massey University [Barczak et al. 2011]. Elas são divididas em 36 categorias ou classes que representam os números de ”0” a ”9” e as letras de ”A” até ”Z”. Exemplos podem ser observados nas Figura 1, Figura 2, Figura 3 e Figura 4. Cada classe possui 70 imagens, com exceção da classe que representa a letra ”T” que tem o total de 65 imagens. Dessa maneira, o conjunto possui um total de 2515 imagens.

As redes foram criadas e treinadas usando o *framework* de aprendizado de máquina de código aberto feito na linguagem Python chamado PyTorch [PyTorch 2020]. Já para a otimização dos hiperparâmetros das redes foi usada a biblioteca para a implementação de algoritmo genético, também escrita em Python, geneticalgorithm [Solgi 2020].

2.1. Pré-processamento

Com o objetivo de reduzir o tempo de treinamento de cada rede neural convolucional foi realizado um pré-processamento nas imagens. Primeiramente, foi usado somente as classes que representam os números de ”0” a ”9”, reduzindo o total de classes de 36 para 10. Após isso, as imagens foram redimensionadas para um tamanho padrão de 32 ”píxeis” de altura e de largura. Por fim, seus 3 canais de cores, *RGB*, foram reduzidos para um único canal, deixando as imagens em preto e branco (Figura 5).

Para o treinamento, as imagens foram divididas em dois grupos. O primeiro grupo, chamado de ”conjunto de treinamento”, é compostos por 80% das imagens. Já o segundo grupo, chamado de ”conjunto de validação”, é formado pelos 20% restantes. A distribuição do conjunto de dados foi feita dessa forma, pois costuma trazer bons resultados. A distribuição pode ser observada na (Tabela 1). Além dessa divisão, cada um dos conjuntos foi subdividido em lotes de tamanho 32, ou seja, até 32 imagens por lote — podendo ter menos, pois o tamanho dos conjuntos não é múltiplo de 32. Esse valor é uma potência de 2 que trouxe bons resultados em testes preliminares. A distribuição dos lotes pode ser vista na (Tabela 2).

2.2. Treinamento

A descrição da rede neural convolucional para o algoritmo genético foi feita em forma de vetor, onde cada posição do vetor contém um sub-vetor de duas posições que representam os valores mínimos e máximos que um hiperparâmetro pode assumir.

Figura. 1. Gesto do número zero

Figura. 2. Gesto do número nove

Conjunto de Treinamento	Conjunto de Validação	Total
560	140	700

Tabela. 1. Distribuição dos conjuntos de imagens após o pré-processamento

Lote do Conj. de Treinamento	Lote do Conj. de Validação	Total de lotes
18	5	23

Tabela. 2. Distribuição dos lotes dentro dos conjuntos de imagens após o pré-processamento

As camadas convolucionais da rede são representadas por 5 sub-vetores. O primeiro descreve o tamanho da camada de saída. O segundo define o tamanho do filtro usado na camada, podendo ser 3, 5 ou 7 (no entanto, o filtro foi limitado para ser usado somente o tamanho 5 — ver seção 4. Dificuldades). Já o terceiro sub-vetor representa se haverá ou não *Batch Normalization* ou Normalização em Lote na camada em questão. O quarto seleciona a função de ativação da camada, que pode ser a função ReLU, ELU ou sigmoide. E por fim, o quinto sub-vetor escolhe qual função de *Pooling* será aplicada, baseado na média ou no máximo. O tamanho do *pooling* para todos os casos é sempre 2, por ser um valor recomendado.

A camada totalmente conectada é representada por 2 sub-vetores. O primeiro sub-vetor descreve o tamanho da camada de saída, enquanto o segundo seleciona a função de ativação, as mesma das camadas convolucionais: ReLU, ELU ou sigmoide.

Figura. 3. Gesto da letra ”A”

Figura. 4. Gesto da letra ”Z”

Os quatro últimos sub-vetores descrevem a função de otimização, taxa de aprendizagem, função de custo e o número de épocas. Existem 4 funções de otimização que podem ser escolhidas pelo algoritmo: *Adagrad* (*Adaptive Gradient*), *SGD* (*Stochastic Gradient Descent*), *RMSprop* (*Root Mean Square Propogation*) e Adam (*Adaptive Moment Estimation*). Já as funções de custo podem ser: *NLLLoss* (*The Negative Log-Likelihood*), *Cross Entropy Loss* e *Multi Margin Loss*.

Os treinamentos foram realizados para gerar redes com 3 camadas de convolução. Esse número foi definido em teste preliminares que mostraram bons resultados. Os treinamentos foram executados usando *hardware* da plataforma Google Colab [Google 2020] por meio de uma conta gratuita. Devido a limitação de tempo de uso do *hardware*, alguns parâmetro do algoritmo genético como, tamanho da população e número de gerações, foram reduzidos para que os treinamentos pudessem ser executados e concluídos. O valor retornado pela função-objetivo para comparação é a taxa de acerto da rede. As configurações do algoritmo podem ser vista na (Tabela 3).

3. Resultados

Foram realizados 12 experimentos no total, onde as rede obtidas trouxeram taxas de acerto muito altas. Esse valor alto pode ser fruto de um sobre-ajuste da rede ao conjunto de treinamento. Os resultados podem ser observados na (Tabela 4).

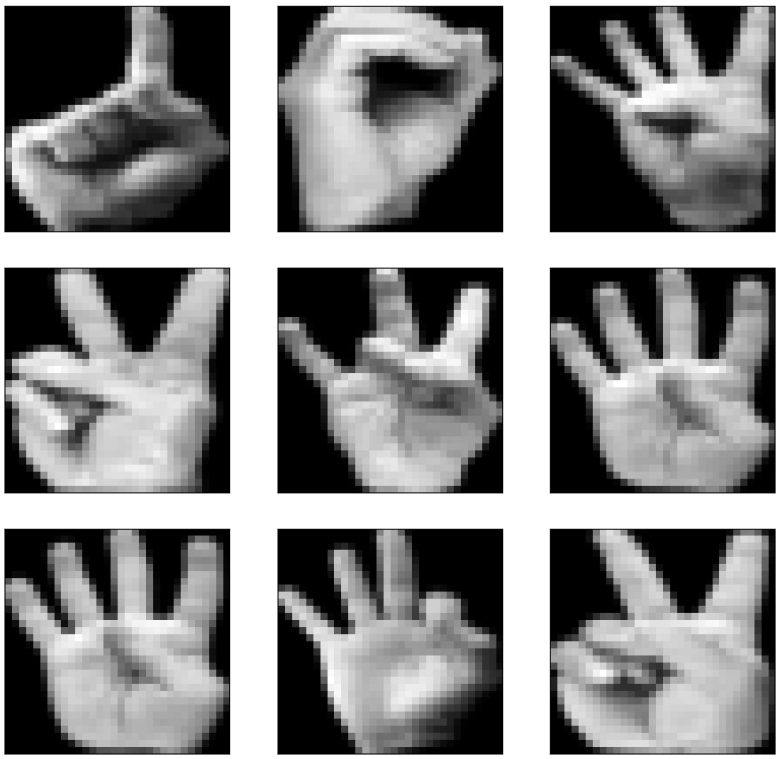

Figura. 5. Imagens após o pré-processamento

Parâmetro	**Valor**
Total de Gerações	20
Tamanho da População	15
Probabilidade de Mutação	10%
Proporção da Elite	1%
Probabilidade de Cruzamento	50%
Tipo de Cruzamento	uniforme
Proporção de Pais	30%

Tabela. 3. Configuração do Algoritmo Genético

Para esse problema de otimização, o algoritmo genético encontrou o resultado ótimo nas primeiras gerações, tendo como pior caso o do nono experimento, que maximizou a função-objetivo na décima geração (o que representa metade do total de gerações).

4. Dificuldades

A principal dificuldade do trabalho foi em relação ao tempo de uso do *hardware* no Google Colab, fazendo com que diversas abordagem fossem tomadas visando diminuir o tempo de treinamento de uma rede. Além disso, algumas abordagens como *Data Augmentation*, o aumento do conjunto de dados por meio de algumas manipulações como rotação ou aplicação de ruído, não puderam ser usadas, o que ajudaria a diminuir a ocorrência de sobre-ajuste. Mesmo com todos aqueles ajustes, algumas execuções foram encerradas prematuramente.

Rede	Taxa de Acerto Treino(%)	Taxa de Acerto Validação(%)
Rede 01	100%	100%
Rede 02	100%	99,29%
Rede 03	100%	98,57%
Rede 04	100%	99,29%
Rede 05	92,68%	95,00%
Rede 06	100%	99,29%
Rede 07	98,21%	100%
Rede 08	100%	100%
Rede 09	100%	100%
Rede 10	98,57%	98,57%
Rede 11	89,46%	85,00%
Rede 12	99,82%	97,14%

Tabela. 4. Taxa de Acerto das Redes Neurais Convolucionais obtidas com o Algoritmo Genético

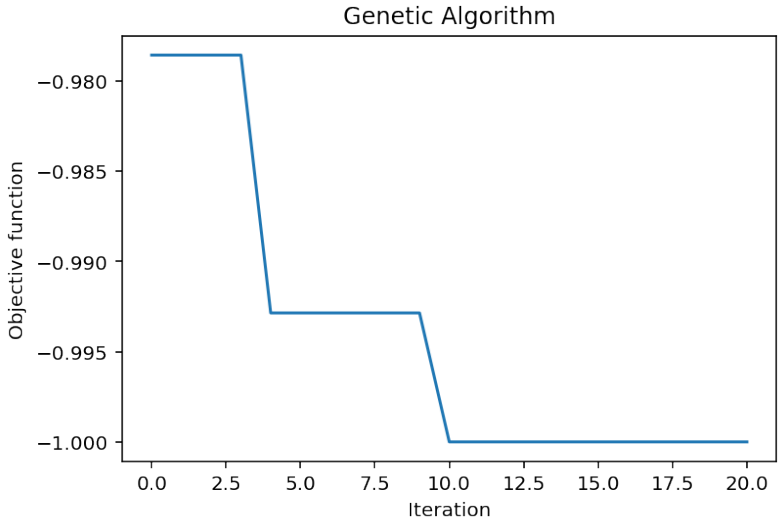

Figura. 6. Valores da função-objetivo no nono experimento

Outro ponto foi o uso de algumas combinações na criação do modelo que apresentavam erro na compilação ou no treinamento. Um exemplo que ocorreu foi o uso de filtro no tamanho 3 ou 7 que encerrava o treino devido a um erro de execução. Com isso, os treinamentos foram limitados ao uso de filtro tamanho 5.

5. Conclusão

O uso de algoritmos genéticos para otimização dos hiperparâmetros de redes neurais convolucionais mostrou bons resultados. Apesar de todas as modificações necessárias para conseguir realizar os experimentos, essa abordagem se mostrou uma ferramenta eficiente para a construção de redes neurais muito complexas.

Os pontos que podem ser abordados em trabalhos futuros são: o uso de *Data*

Augmentation para expandir o tamanho do conjunto de dados, não reduzir tanto o tamanho das imagens como foi feito na etapa 2.1. Pré-processamento, bem como, não reduzir os canais de cores, além de usar todas as 36 classes disponíveis no conjunto de dados. Já para o algoritmo genético, seria interessante aumentar o número de gerações e o tamanho da população, assim como ajustar a função-objetivo para penalizar resultados de redes sobre-ajustadas.

References

[Barczak et al. 2011] Barczak, A. L. C., Reyes, N. H., Abastillas, M., Piccio, A., and Susnjak, T. (2011). A new 2d static hand gesture colour image dataset for asl gestures. *Research Letters in the Information and Mathematical Sciences*, 15:12–20.

[Deb 2001] Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons.

[DeepAI 2020] DeepAI (2020). Hyperparameter definition — deepai. `https://deepai.org/machine-learning-glossary-and-terms/hyperparameter`. Accessed: 2020-11-03.

[Google 2020] Google (2020). Google colaboratory. `https://colab.research.google.com/`. Accessed: 2020-11-03.

[PyTorch 2020] PyTorch (2020). Pytorch: An open source machine learning framework. `https://pytorch.org/`. Accessed: 2020-11-03.

[Rosenblatt 1957] Rosenblatt, F. (1957). The perceptron — a perceiving and recognizing automaton.

[Sharma 2020] Sharma, V. (2020). Deep learning – introduction to convolutional neural networks — vinod sharma's blog. `https://vinodsblog.com/2018/10/15/everything-you-need-to-know-about-convolutional-neural-networks/`. Accessed: 2020-11-03.

[Solgi 2020] Solgi, R. M. (2020). geneticalgorithm is a python library. `https://github.com/rmsolgi/geneticalgorithm`. Accessed: 2020-11-03.