Escola de Engenharia da UFMG

Trabalho Final de Otimização

Otimização da Arquitetura de Redes Neurais com Algoritmos Genéticos

Aluno: Giovanni Martins de Sá Júnior - 2017001850

Belo Horizonte, 28 de agosto de 2024

Introdução

Os algoritmos genéticos são métodos de otimização na teoria da Evolução Natural, proposta por Charles Darwin. Neste método, são simulados processos evolutivos como seleção, cruzamento, e mutação para encontrar possíveis soluções ótimas para problemas de grande complexidade. Assim, esses algoritmos funcionam tal qual uma seleção natural, em que uma população compete entre si, e os mais aptos têm maiores chances de passar as suas características para as próximas gerações.

Já as redes neurais, são modelos computacionais inspirados na estrutura do cérebro humano, composto por camadas de neurônios artificiais que processam as informações e aprendem padrões a partir de dados. Um famoso modelo conhecido, é o Perceptron de Multicamadas, que é projetado por diferentes camadas conectadas entre si, contendo neurônios, onde cada neurônio está conectado a todos os neurônios da camada seguinte. Assim, a arquitetura é eficiente para problemas de regressão e classificação, sendo capaz de entender padrões complexos nos dados.

Diante disso, a integração de algoritmos genéticos com as redes neurais podem trazer melhorias significativas na arquitetura e no desempenho dessas redes. Tradicionalmente, a construção e a otimização de redes neurais, como a escolha do número de camadas, a quantidade de neurônios por camada e os hiperparâmetros são definidos por meio de tentativa e erro, ou por técnicas baseadas em gradiente. Com isso, os algoritmos genéticos podem oferecer uma abordagem alternativa para a definição de uma arquitetura mais eficiente na resolução do problema proposto.

Assim sendo, ao se aplicar algoritmos genéticos para ajustar os parâmetros de uma MLP, pode ser possível alcançar uma configuração para o modelo que inicialmente não pudesse ser obtido por métodos convencionais. Deste modo, a combinação de ambas as técnicas pode permitir a criação de modelos mais eficientes e robustos.

Objetivos do Trabalho

- Encontrar uma arquitetura ótima para uma rede neural MLP(Multi Layer Perceptron), utilizando algoritmos genéticos, a partir de uma base de dados definida;
- Comparar os resultados de performance de uma arquitetura otimizada pelo algoritmo genético com uma arquitetura não otimizada e com os parâmetros pré-definidos.

Algoritmos Genéticos

Os algoritmos genéticos são técnicas de otimização baseados nos princípios da evolução natural, simulando o processo de seleção natural para encontrar soluções ótimas em problemas complexos. Assim o algoritmo genético pode ser dividido em diferentes etapas:

- **Inicialização:** definição de uma população inicial de possíveis soluções (para este trabalho, indivíduos, mas poderiam ser cromossomos, por exemplo) é inicializada de maneira aleatória. Cada solução irá ser representada por um vetor de números, que correspondem aos parâmetros do problema a ser otimizado.
- **Avaliação:** cada indivíduo da população será avaliado por uma função de aptidão que irá medir a qualidade da solução em relação ao problema. Assim a função de aptidão determina quão bem a solução resolve o problema, atribuindo uma pontuação a cada indivíduo.
- **Seleção:** Os indivíduos mais aptos possuem uma maior probabilidade de serem solucionados para a reprodução. Deste modo, por meio de métodos de seleção, um subconjunto de soluções é selecionado aleatoriamente e a melhor dentre elas é escolhida.
- **Reprodução:** Novos indivíduos (soluções) são criados a partir das soluções selecionadas utilizando operadores genéticos como cruzamento e mutação. O cruzamento combina genes de duas soluções e gera uma nova solução, enquanto a mutação altera aleatoriamente um ou mais genes de uma solução. Por meio destas duas técnicas é possível aumentar a diversidade da população.
- **Substituição:** As novas soluções substituem parte ou todas as soluções da população geral formando uma nova geração.
- **Iteração:** O processo de avaliação, seleção, reprodução e substituição é repetido por diferentes gerações até que uma solução suficientemente boa seja encontrada, por meio de um critério de parada. Para este trabalho, o critério de parada vai ser delimitado pelo número de

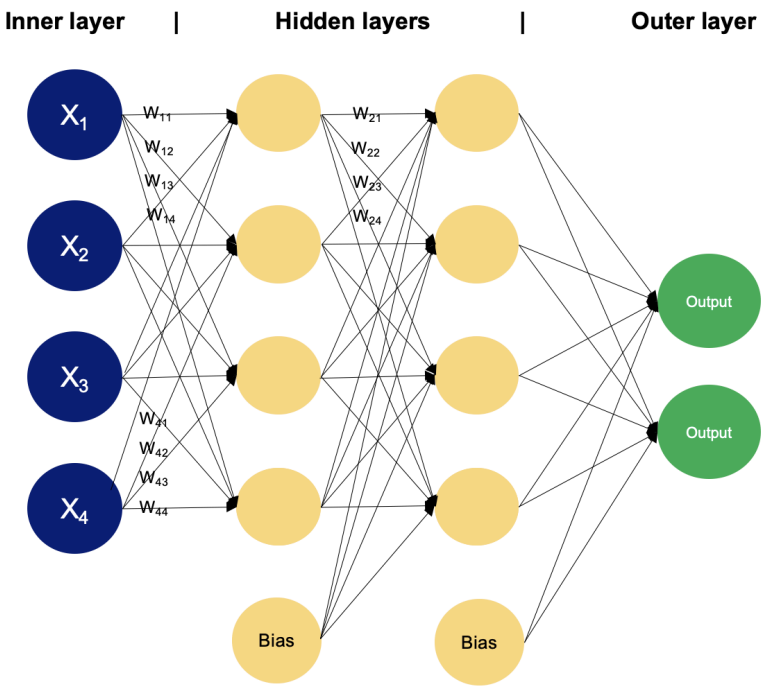
Assim, o objetivo final dos algoritmos genéticos é fazer com que a população evolua com o passar das gerações, em direção a uma solução ótima para o problema proposto. A combinação das etapas anteriores permite que o algoritmo explore e refine o espaço de soluções de forma eficiente.

Multi Layer Perceptron

As redes neurais são modelos computacionais inspirados no cérebro humano, compostos por neurônios artificiais que processam os padrões e aprendem a partir dos dados. O Multilayer Perceptron é um dos modelos mais conhecidos, consistindo de múltiplas camadas de neurônios totalmente conectados.

De maneira geral, a MLP apresenta três tipos de camadas, uma camada de entrada, uma ou mais camadas ocultas, e uma camada de saída. Cada neurônio de cada camada está conectado com todos os neurônios da camada de saída, o que forma uma densa camada de conexões. Assim, cada conexão entre neurônios irá apresentar um neurônio associado, determinando a importância do sinal transmitido. Durante a etapa de treinamento, os pesos são ajustados pelo processo de retropropagação, minimizando a diferença entre a saída prevista e a saída desejada por meio do gradiente descendente.

Figura 1 - Arquitetura de uma MLP



Acesso em: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning

As camadas ocultas permitem que a rede capture e aprenda padrões não lineares, tornando eficazes em problemas de classificação por exemplo, foco do desenvolvimento deste trabalho.

Metodologia

Para o desenvolvimento deste trabalho, será feita a análise comparativa do desempenho de modelos de Multilayer Perceptron antes e depois da otimização de arquitetura com algoritmos genéticos, com a utilização de diferentes bases de dados.

Assim, serão levados em conta, para fins de análise deste trabalho, a acurácia dos modelos em executar a classificação, o comportamento das curvas da função de custa à medida que as épocas são executadas, bem como o tempo de treinamento envolvido em todo o processo. Por fim, será plotada uma matriz de confusão a partir das inferências feitas a partir das inferências feitas pelos modelos

Abaixo, serão discutidos em detalhes informações mais a respeito da metodologia:

Bases de Dados escolhidas

MNIST

O dataset MNIST apresenta um acervo de imagens digitalizadas de números escritos à mão, possuindo dez classes distintas (números de zero a nove). O conjunto de imagens é dividido em sessenta mil imagens de treinamento e dez mil imagens de teste, de resolução 28x28 pixels.

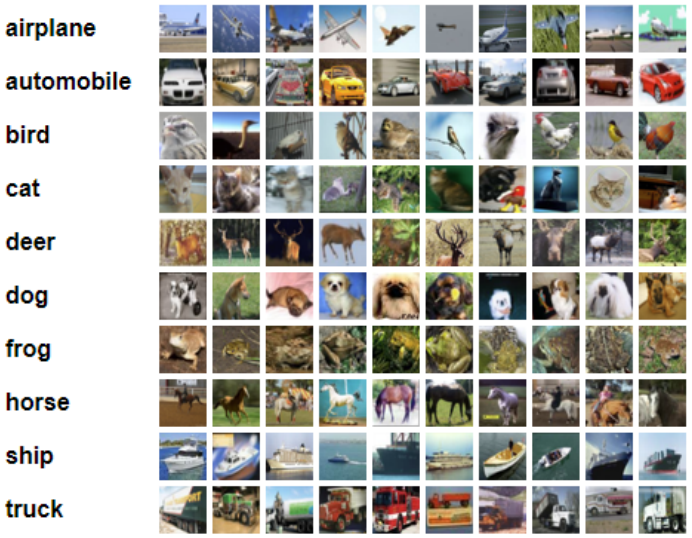
Figura 2 - Amostra do dataset MNIST

Acesso em: https://en.wikipedia.org/wiki/MNIST_database

CIFAR-10

O conjunto CIFAR-10 apresenta um acervo de imagens de animais e veículos, conforme pode ser visto na figura abaixo, com o sufixo de dez representando o número de classes presentes no dataset. O conjunto em si apresenta sessenta mil imagens de resolução 32x32 pixels, sendo cinquenta mil imagens de treinamento, e dez mil imagens de teste. Há também uma versão estendida deste conjunto de imagens, conhecida por CIFAR-100.

Figura 3 - Amostra do dataset CIFAR-10

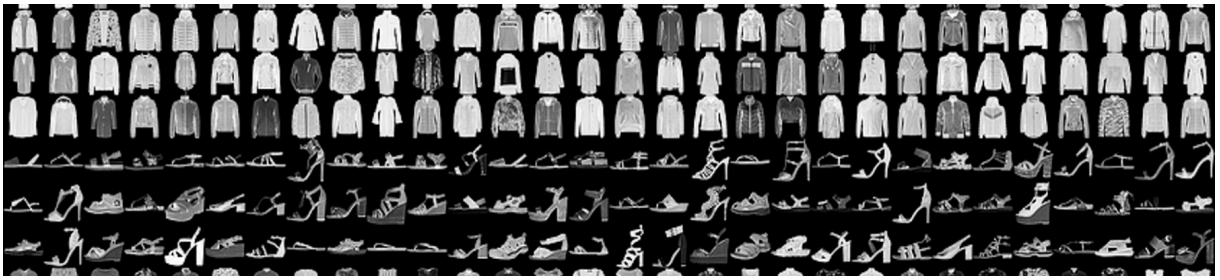


Acesso em: https://en.wikipedia.org/wiki/MNIST_database

FASHION-MNIST

O dataset FASHION-MNIST apresenta um conjunto de imagens em escala de cinza, representando itens relacionados ao vestuário, como vestidos, camisetas, sapatos, bolsas, tênis, dentre outros, resultando em dez classes distintas. O conjunto apresenta sessenta mil imagens de treinamento, e dez mil imagens de teste, em resolução 28x28 pixels.

Figura 3 - Amostra do dataset FASHION-MNIST



Acesso em: https://www.kaggle.com/datasets/zalando-research/fashionmnist

Implementação do Algoritmo Genético

Nesta etapa, será discutido o funcionamento do algoritmo genético, para isso um indivíduo seŕá composto por quatro “genes” distintos: número de neurônios, índice da função de ativação (sendo zero para representar a função ReLU, e um para representar a função Sigmoidal), a taxa de aprendizado (utilizada para o treinamento da rede), e o tamanho do batch a ser processado pela rede.

Assim, uma rede será treinada por três épocas por meio de uma função de avaliação em que será calculado a acurácia(sendo aqui definida pelo *fitness*) a partir do “genes do indivíduo”. Dependendo do indivíduo, ele pode ou não permanecer na população para passar pelos processos do algoritmo genético como a mutação, o cruzamento e a seleção. Assim o procedimento será repetido por cinco gerações até que o melhor indivíduo seja encontrado. Abaixo, é mostrada a implementação executada:

Figura 4 - Função de avaliação do indivíduo a ser utilizada no algoritmo genético

```python
# Função de avaliação do indivíduo
def evaluate_individual(individual):
    n_neurons, activation_idx, learning_rate, batch_size = individual

    # Certifique-se de que activation_idx esteja dentro do intervalo esperado
    activation_idx = min(max(activation_idx, 0), 1)
    activation_fn = [nn.ReLU(), nn.Sigmoid()][activation_idx]

    model = SimpleNet(n_neurons, activation_fn)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)

    # Treinamento no conjunto de treinamento
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)
    for epoch in range(3):  # Treinando por 3 épocas como exemplo
        for inputs, labels in trainloader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

    # Avaliação no conjunto de validação
    validloader = torch.utils.data.DataLoader(validset, batch_size=batch_size, shuffle=False)
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in validloader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = correct / total
    return accuracy,
```

Figura 5 - Implementação principal do algoritmo genético

```
# Configuração do Algoritmo Genético
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_int", random.randint, 10, 200)  # Número de neurônios
toolbox.register("attr_activation", random.randint, 0, 1)  # Função de ativação (ReLU ou Sigmoid)
toolbox.register("attr_float", random.uniform, 0.001, 0.1)  # Taxa de aprendizado
toolbox.register("attr_batch", random.randint, 32, 128)  # Tamanho do batch

toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_int, toolbox.attr_activation, toolbox.attr_float, toolbox.attr_batch), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", evaluate_individual)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=10, up=200, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# Algoritmo Genético
population = toolbox.population(n=10)
NGEN = 5
for gen in tqdm(range(NGEN), desc='Gerações'):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.2)

    fits = list(tqdm(map(toolbox.evaluate, offspring), total=len(offspring), desc="Avaliando Indivíduos"))

    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population = toolbox.select(offspring, k=len(population))

# Seleciona o melhor indivíduo
top_ind = tools.selBest(population, k=1)[0]

# Parâmetros do melhor indivíduo
best_params = {
    'n_neurons': top_ind[0],
    'activation_idx': top_ind[1],
    'learning_rate': top_ind[2],
    'batch_size': top_ind[3]
}
```

Arquitetura da MLP

Para a arquitetura do Rede Neural utilizada, foi projetada um Multilayer Perceptron a partir da biblioteca chamada PyTorch, sendo definidas uma camada de entrada dos dados, uma camada intermediária que irá receber um número de neurônios da camada de saída e a função de ativação escolhida do melhor indivíduo, e a camada de saída que irá ser ajustada pela base de dados utilizada. Assim, se a base de dados escolhida apresentar dez classes, serão definidos dez neurônios para a camada de saída. No exemplo abaixo, a camada intermediária é representada pela variável **fc1**, enquanto a camada de saída, pela variável **fc2**.

Figura 6 - Arquitetura do Multilayer Perceptron

```python
# Definindo a arquitetura da rede neural
class SimpleNet(nn.Module):
    def __init__(self, n_neurons, activation_fn):
        super(SimpleNet, self).__init__()
        self.fc1 = nn.Linear(28 * 28, n_neurons)
        self.activation = activation_fn
        self.fc2 = nn.Linear(n_neurons, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = self.fc1(x)
        x = self.activation(x)
        x = self.fc2(x)
        return x
```

Treinamento dos modelos de MLP:

Após a escolha do melhor indivíduo pelo o algoritmo genético, serão efetuados quatro treinamentos de redes neurais, três com redes não otimizadas e com parâmetros pré-definidos, e outra com os parâmetros definidos pelo melhor indivíduo do algoritmo genético. Assim, ambas serão treinadas por dez épocas, terão as funções de custo, acurácias de classificação, e tempo de execução de treinamento para fins de comparação. Por fim, uma matriz de confusão será plotada visando analisar a classificação dos dados dos respectivos conjuntos de teste, definidos por cada dataset.

Resultados

Nesta seção, serão apresentados os resultados obtidos após o treinamento e teste para os quatro modelos definidos, a partir do dataset utilizado. Mais abaixo, são apresentadas duas tabelas, uma apresentando as características de cada modelo, desde o número de neurônios da camada intermediária, a função de ativação escolhida e a taxa de aprendizado. Para a segunda tabela, é apresentada uma tabela contendo a performance de cada modelo, como a acurácia, o custo final de treinamento de cada modelo, e o tempo envolvido no processo.

MNIST:

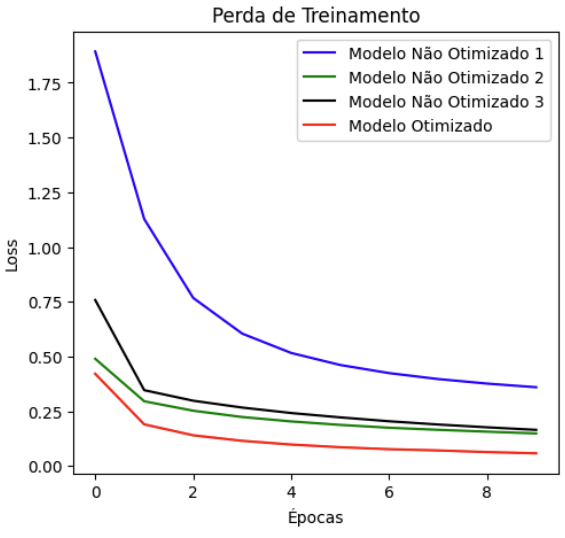
Tabela 1 - Comparação de Arquitetura entre Modelos - MNIST

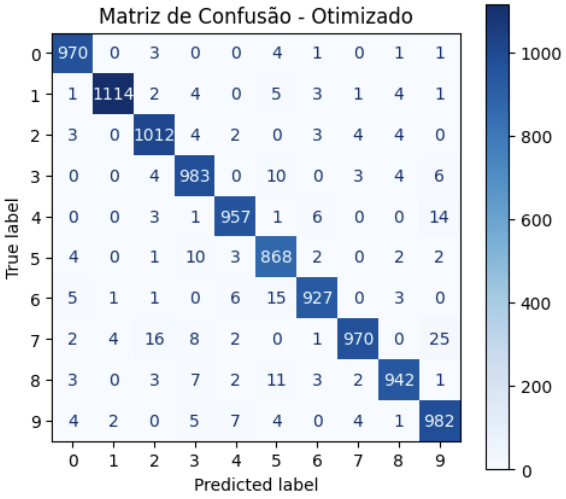
	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
N° neurônios Cam. Oculta	50	20	150	86
Função de Ativação	Sigmoidal	ReLu	Sigmoidal	ReLu
Taxa de Aprendizado	0.01	0.05	0.07	0.0961

Tabela 2 - Comparação de Performance entre Modelos - MNIST

	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
Acurácia	0.9087	0.9484	0.9519	0.9725
Loss (J)	0.3601	0.1491	0.1655	0.0584
Tempo gasto	161.49	160.61	168.47	164.74

Figuras 7 e 8 - Funções de Custo calculadas para os modelos e matriz de confusão do modelo otimizado





FASHION-MNIST:

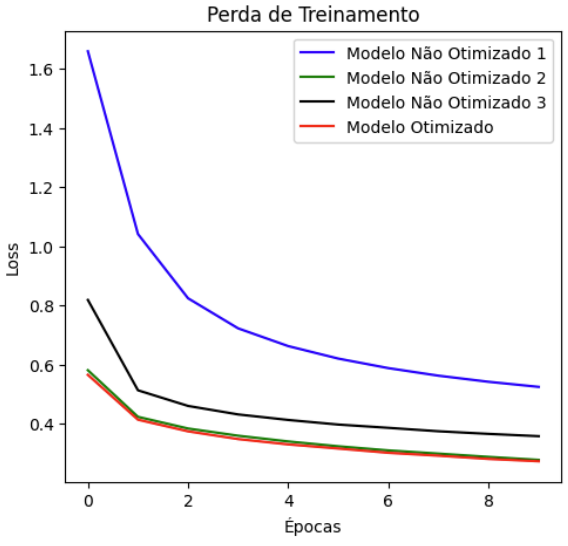
Tabela 3 - Comparação de Arquitetura entre Modelos - FASHION MNIST

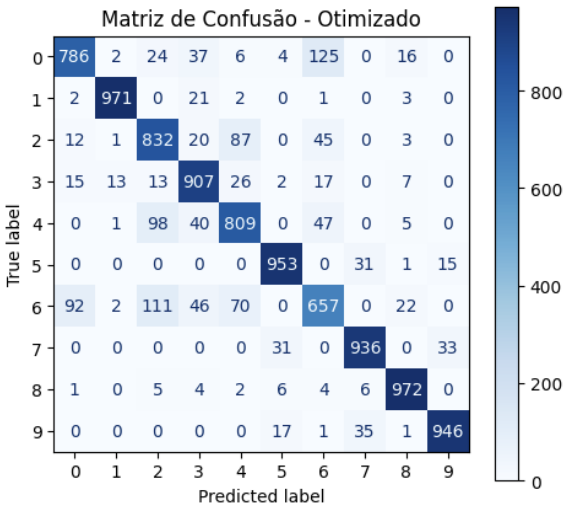
	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
N° neurônios Cam. Oculta	50	200	150	95
Função de Ativação	Sigmoidal	ReLU	Sigmoidal	ReLU
Taxa de Aprendizado	0.01	0.05	0.07	0.0778

Tabela 4 - Comparação de Performance entre Modelos - FASHION MNIST

	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
Acurácia	0.8056	0.8626	0.8500	0.8769
Loss (J)	0.5252	0.2786	0.3587	0.2739
Tempo gasto (s)	172.31	165.15	169.91	169.92

Figuras 9 e 10 - Cálculo da função de custo com o passar das épocas





CIFAR-10 - MLP

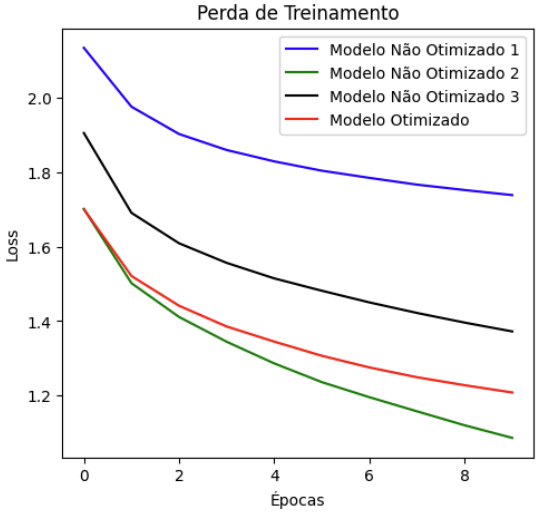
Tabela 5 - Comparação de Arquitetura entre Modelos - CIFAR 10

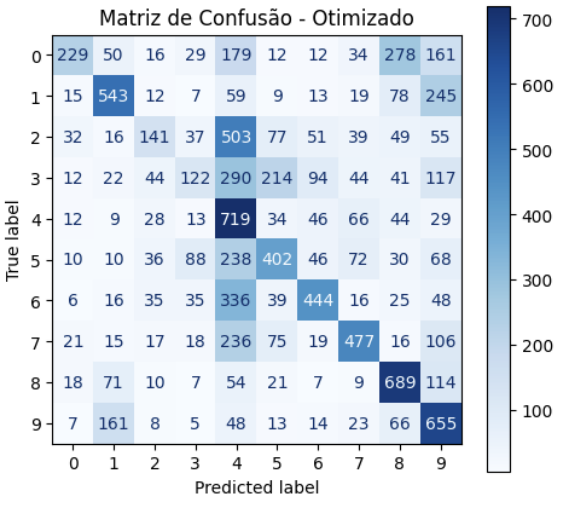
	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
N° neurônios Cam. Oculta	50	200	150	´72
Função de Ativação	Sigmoidal	ReLU	Sigmoidal	ReLU
Taxa de Aprendizado	0.01	0.05	0.07	0.068

Tabela 6 - Comparação de Performance entre Modelos - CIFAR 10

	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
Acurácia	0.3972	0.4848	0.4960	0.4421
Loss (J)	1.7385	0.4960	1.3725	1.2085
Tempo gasto (s)	162.50	186.57	171.88	164.18

Figuras 11 e 12 - Cálculo da função de custo com o passar das épocas





CIFAR 10 - CNN (Rede Neural Convolucional)

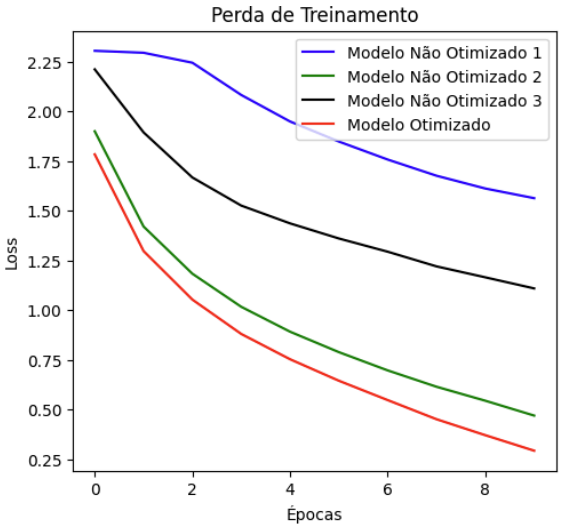
Tabela 7 - Comparação de Arquitetura entre Modelos - CIFAR 10

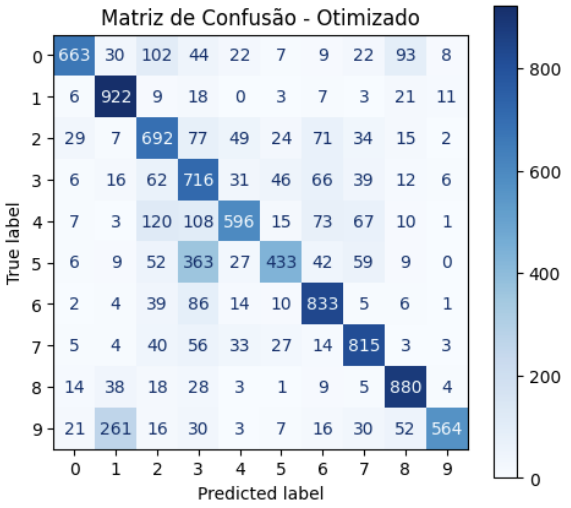
	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
N° neurônios Cam. Oculta	50	200	100	174
Função de Ativação	Sigmoidal	ReLU	Sigmoidal	ReLU
Taxa de Aprendizado	0.01	0.05	0.07	0.0911

Tabela 8 - Comparação de Performance entre Modelos - CIFAR 10

	Modelo Não Otimizado 1	Modelo Não Otimizado 2	Modelo Não Otimizado 3	**Modelo Otimizado**
Acurácia	0.4367	0.7010	0.5828	0.7114
Loss (J)	1.5638	0.4696	1.1095	0.2929
Tempo gasto (s)	1223.99	1337.79	961.61	905.20

Figuras 11 e 12 - Cálculo da função de custo com o passar das épocas





Conclusão:

Como foi possível obter nos testes, os algoritmos genéticos ofereceram uma abordagem interessante para otimização de redes neurais, principalmente quando a busca por hiperparâmetros e arquitetura se torna difícil de ser estabelecida por meios tradicionais.

Uma vantagem significativa dos algoritmos genéticos, têm relação com a sua capacidade de conseguir escapar de mínimos locais, o que pode ser um desafio para algoritmos baseados em gradientes, principalmente na busca de espaços não convexos. Por outro lado, a utilização dos AG's podem ser computacionalmente intensivos, das múltiplas avaliações de desempenho para as diferentes configurações das redes, exigindo muito tempo de processamento, principalmente para grandes volumes de dados.

Por fim, os algoritmos genéticos são uma valiosa ferramenta para a otimização da arquitetura de redes neurais , se tornando uma alternativa interessante a métodos mais tradicionais, explorando um maior espaço de soluções com a ressalva do custo computacional maior.