RECONHECIMENTO DE SIMBOLOS MUSICAIS: UMA APLICAÇÃO DA BIBLIOTECA PYTORCH

Gabriel Augusto Neto[1], Sandra Cristina Costa Prado[2]

[1] Discente do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas Gabriel.augusto@fatec.sp.gov.br

[2] Docente do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas sandra.prado01@fatec.sp.gov.br

RESUMO

Neste trabalho, são mostrados aspectos teóricos e práticos da implementação de uma rede neural convolucional aplicado à tarefa de classificação de imagens manuscritas de símbolos utilizados na escrita musical ocidental. Foi utilizada a biblioteca pytorch, apropriada para implementar redes neurais profundas. A rede neural proposta apresenta três camadas convolucionais, ativadas pela função ReLu e seguidas, respectivamente, de operações de Pooling. Ao fim da rede foram utilizadas camadas completamente conectadas com uma saída ativada pela função Softmax. Para realizar o aprendizado supervisionado da rede neural convolucional descrita, foi utilizado o dataset HOMUS que apresenta símbolos musicais escritos manualmente por estudantes de música. Após o treinamento foi constado que o modelo não conseguiu aprender a classificar as imagens. Acredita-se que a escassez de dados supervisionados foi a causa do não-aprendizado.

Palavras-chave: Reconhecimento de Imagens; Redes Neurais Convolucionais; Pytorch

1 INTRODUÇÃO

Neste trabalho, foram percorridas as etapas necessárias para um problema de reconhecimento de imagens utilizando redes neurais convolucionais implementadas com a biblioteca Pytorch. Também foram utilizados diversos pacotes da linguagem Python para facilitar e agilizar o desenvolvimento do código.

Os dados processados na implementação das redes neurais convolucionais são armazenados em uma estrutura de dados especial da biblioteca Pytorch, chamada de tensor, que possui várias funcionalidades. As que mais se destacam são as funcionalidades implementadas na linguagem CUDA, específica para hardwares do

tipo GPU. Esta característica torna o processamento dos dados muito mais veloz e possibilita a fase de treinamento das redes neurais profundas que tem alta demanda computacional.

A implementação foi feita no ambiente de desenvolvimento integrado (IDE) Google Colab. disponível online através de uma conta google. É acessado através de qualquer navegador, sua utilização parcialmente gratuita e não há necessidade de configuração do ambiente e importação das bibliotecas pois a plataforma funciona diretamente nos servidores da Google. Seu sistema de gerenciamento de arquivos é atrelado a uma conta Google e os projetos são salvos no Google Drive. O Colab utiliza princípios de funcionamento baseados no sistema operacional Linux, possuindo muitos dos comandos, principalmente àqueles voltados para manipulação de pastas e arquivos, similares a comandos utilizados no terminal Linux, como !pip install e !pwd.

O dataset de símbolos musicais utilizado neste projeto foi o Homus – Handwritten Online Music Symbols (HOMUS, 2021), que possui 15.200 imagens distribuídas não uniformemente entre 32 símbolos musicais. Os símbolos foram escritos à mão por cem diferentes músicos, cada um com seu estilo próprio de escrita.

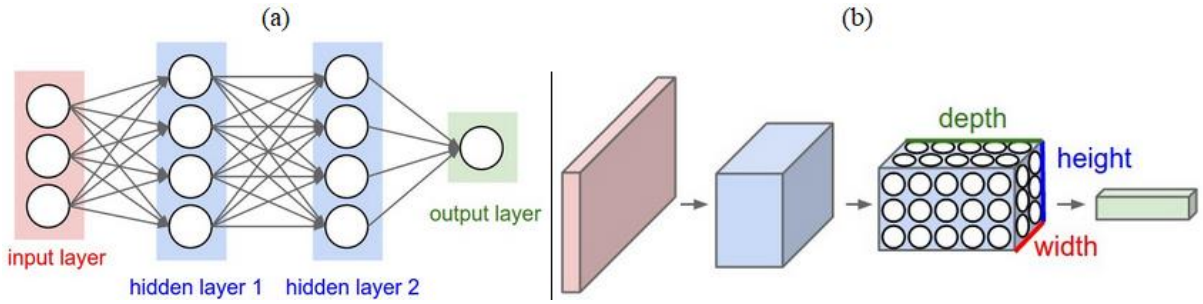
2 REFERENCIAL TEÓRICO)

Uma rede neural convolucional – CNN (Convolutional Neural Network)) é um algoritmo de redes neurais que apresenta uma forma otimizada de conexão entre camadas ocultas de neurônios.

A figura 1 (a) é uma representação de uma rede neural convencional, na qual cada neurônio de uma camada se conecta com todos os neurônios da camada anterior e da seguinte. Entre a primeira camada, input layer, e a última, output layer, há uma ou várias camadas ocultas. As conexões entre camadas adjacentes, representadas por setas, são parâmetros dos neurônios chamados pesos que definem a importância de cada entrada e têm seus valores determinados no processo de treinamento da rede, a partir dos dados de entrada. Cada neurônio possui, ainda, um outro parâmetro chamado bias (pronuncia-se “baias”), também determinado no treinamento. A quantidade de camadas e de neurônios e a forma como se conectam definem a arquitetura da rede neural.

Na arquitetura de CNN, os pesos que conectam as camadas são compartilhados. Este fato faz com que a CNN apresente uma arquitetura própria e com muito menos parâmetros (pesos e bias) quando comparada com a arquitetura de uma rede neural convencional. A figura 1 (b) ilustra o esquema de camadas convolucionais em uma CNN. Cada volume tridimensional – 3D representa uma camada convolucional, sendo que o primeiro volume (rosa) corresponde aos dados de entrada. A profundidade (depth) está relacionada à quantidade de canais que a camada pode apresentar. Volumes com uma profundidade maior indicam que foram usados muitos canais naquela camada. À medida que as convoluções são realizadas, a largura e altura dos volumes se tornam cada vez menores.

Figura 1 – (a) Rede neural convencional com três camadas. (b) Rede neural convolucional (CNN): organiza seus neurônios em três dimensões (largura, altura, profundidade (width, height, depth)). Cada camada de uma CNN transforma o volume de entrada tridimensional – 3D em um volume de saída 3D de neurônios. Neste esquema, a camada de entrada vermelha contém uma imagem. Sua largura e altura são as dimensões da imagem e a sua profundidade são os canais vermelho, verde, azul, caso a imagem seja colorida.



Fonte: CS231n, https://cs231n.github.io/convolutional-networks/, 2021.

Devido à sua arquitetura ser baseada no funcionamento neuro-visual humano, as CNNs são uma das arquiteturas mais apropriadas para o reconhecimento de imagens por meio de um algoritmo de ML. Além disso, devido à sua arquitetura não necessitar de muitos pesos entre camadas ocultas de neurônios, a CNN possibilita projetar várias camadas ocultas a fim de realizar aprendizagem profunda (*deep learning),* levando a alta performance em tarefas de classificação de imagens.

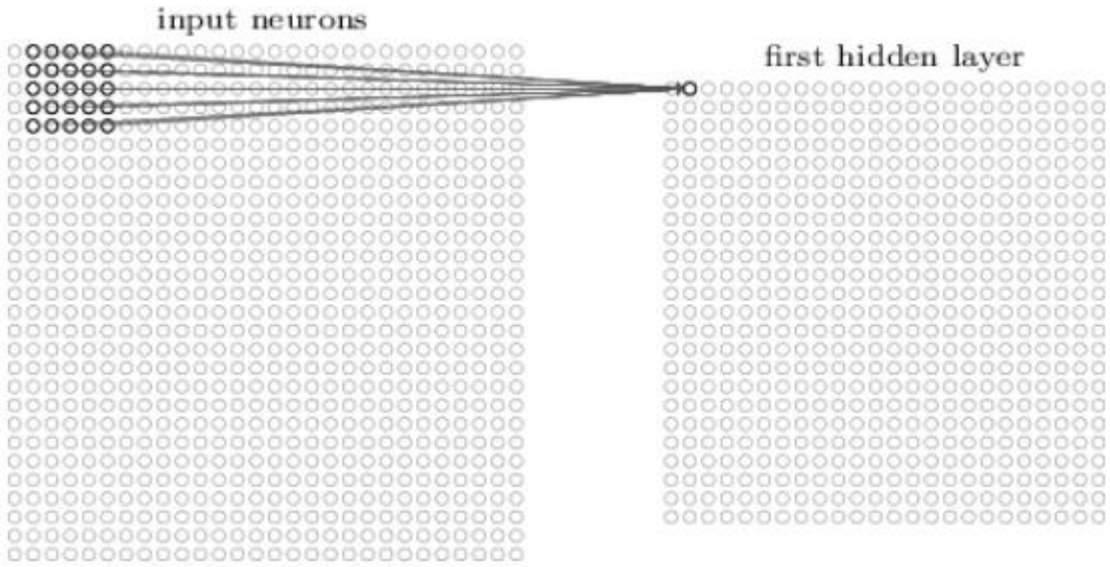
Enquanto em redes neurais convencionais, os pixels de uma imagem são dispostos em um array 1D, nas CNNs a imagem é mantida com seu formato 2D

original. Portanto, as unidades de entrada em uma CNN são arrays 2D que correspondem aos valores de cada pixels que a imagem possui.

A camada oculta de neurônios em uma CNN, também apresenta a forma de array 2D e tem o nome de feature-map. Sendo que a conexão entre as unidades de entrada e a camada oculta se dá por um conjunto de pesos pequeno. Este conjunto forma o filtro ou kernel de convolução. Os valores dos pesos que compõe o filtro também são dispostos em uma array 2D. Na grande maioria dos casos, o filtro tem tamanho que varia de 3x3 até 5x5 pesos, podendo ser também na forma retangular (3x4 pesos, por exemplo).

A partir do filtro é realizado o que se denomina como convolução. A operação de convolução sobre as unidades de entrada (pixels da imagem em array 2D) é esquematizada na figura 2. A convolução consiste em multiplicar os valores de entrada de um trecho 2D com mesmo tamanho do filtro, pelos respectivos valores de peso que compõe o filtro. Este trecho da imagem com o mesmo tamanho do filtro é chamado de campo receptivo local.

Figura 2 – Diagrama mostrando a conexão entre unidades de entrada (campo receptivo local) e neurônio da camada oculta. As setas correspondem aos pesos que compõem o filtro.

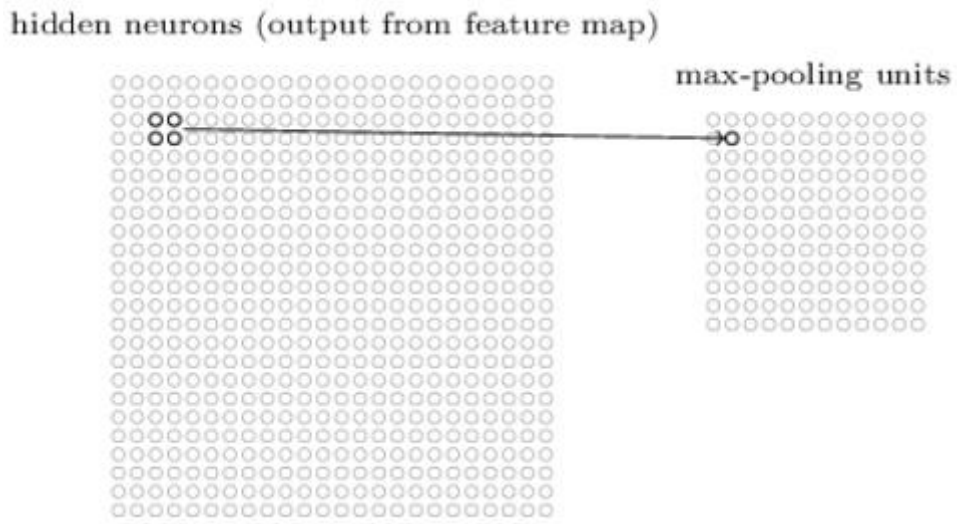


Fonte: Nielsen, 2021, http://neuralnetworksanddeeplearning.com/chap6.html.

Depois da convolução deve-se aplicar alguma função de ativação sobre o resultado guardado em cada unidade da camada oculta. Neste trabalho, aplicou-se a função ReLu que é amplamente utilizada.

Na sequência, foi realizada uma simplificação sobre os dados na camada oculta de tal forma que há uma redução das arrays 2D. Esta operação é chamada de pooling e não é obrigatória, entretanto, é comum utilizá-la para diminuir a quantidade de neurônios entre camadas ocultas. A figura 3 ilustra uma operação de pooling onde é feita uma redução de cada trecho 2x2 da camada oculta. Ou seja, cada trecho com 2x2 neurônios é reduzido a um único neurônio, conforme mostra a figura. É possível fazer a redução de várias formas, sendo que o procedimento mais comum é o Max-Pooling, onde o maior valor dentro da região 2x2 é escolhido para o reduzir.

Figura 3 – Diagrama mostrando a operação de MaxPooling. Da região 2x2 de neurônios é escolhido o neurônio que apresenta o maior valor. A seta indica que os quatro neurônios foram substituídos pelo que apresenta maior valor.



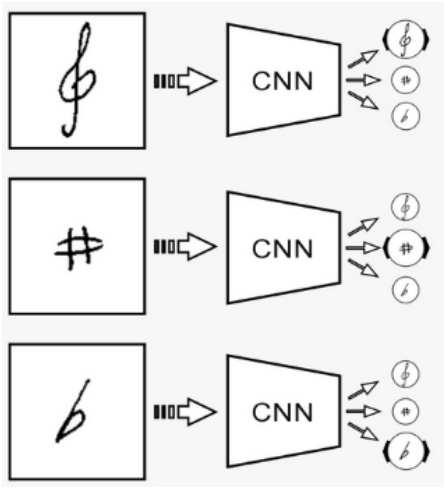
Fonte: Nielsen, 2021, http://neuralnetworksanddeeplearning.com/chap6.html.

Uma rede convolucional, em geral, apresenta sucessivas camadas convolucionais seguidas da operação pooling. Ao final de todo este processo, os neurônios resultantes são dispostos em uma array 1D e se conectam completamente com uma nova camada e, a partir daí, serão processados até que se obtenha as saídas esperadas.

A quantidade de neurônios da última camada corresponde à quantidade de símbolos (classes) que a rede se propõe a classificar. A classificação das imagens ocorre após a rede passar por um procedimento chamado fase de treinamento. Após esta fase, a imagem a ser classificada é recebida como dado de entrada e processada pela CNN que retornará uma saída com a classificação da imagem. A resposta é

determinada pela ativação do neurônio de saída correspondente ao símbolo da imagem de entrada. Este processo é ilustrado pela figura 4.

Figura 4 – Diagrama representando como a CNN classifica as imagens. Se a imagem recebida na entrada é uma clave de Sol, o primeiro neurônio de saída deve ser o mais ativado. Se a imagem recebida na entrada é um sustenido o segundo neurônio será o mais ativado. Se a imagem de entrada é um bemol, o terceiro neurônio de saída será o mais ativado.



Fonte: Elaborada pelos autores.

3 METODOLOGIA

Um projeto de *Machine Learning* consiste em diversas fases. Dentre elas, há três fases sequenciais que se destacam: a fase da preparação dos dados, a fase do treinamento do algoritmo de ML e a fase de teste do modelo já treinado.

3.1 Preparação dos Dados

O algoritmo de rede neural, assim como todos os algoritmos de ML, necessita de dados para seu funcionamento. Portanto, os dados devem ser preparados antes de qualquer outra tarefa.

Na preparação dos dados é preciso verificar se as informações do *dataset* estão balanceadas, ou seja, se a quantidade de imagens é uniforme para cada símbolo musical que a rede deve identificar. Se o dataset não estiver balanceado, a rede neural terá eficiência menor para identificar os símbolos com menos exemplos disponíveis

na fase de treinamento. Felizmente, o dataset utilizado neste projeto é balanceado, ou seja, ele contém quantidade uniforme de exemplos para cada símbolo musical.

A próxima etapa, nesta fase, é reservar dados para o treinamento e dados para a fase de teste. Portanto, separa-se dados em duas partes: uma a ser utilizada no treinamento que é chamada de dados de treinamento e outra para a avaliação da performance do modelo que é chamada de dados de teste. O tamanho dos conjuntos de treinamento e de teste pode variar, mas vale ressaltar que para o treinamento é reservado uma maior parcela dos dados. É necessário realizar esta separação pois com os dados de treinamento a rede neural irá encontrando seus melhores parâmetros de forma iterativa afim de minimizar os erros na classificação das imagens. Em contrapartida para testar a capacidade preditiva do modelo deve-se usar os dados de teste, não vistos pelo modelo em seu treinamento.

O dataset HOMUS apresenta as imagens separadas em diretórios exclusivos para cada símbolo musical. Dos trinta e dois símbolos disponíveis neste dataset, foram utilizados dezesseis. Portanto, a rede convolucional montada será um classificador para as dezesseis possíveis classes.

Para realizar a separação dos dados de treinamento e teste foi utilizado um módulo da linguagem Python chamado *Splitfolders.* Neste módulo há duas funções para separar os dados: a função *ratio(),* utilizada quando o *dataset* já se encontra balanceado; o segundo método é o *fixed()*, quando os dados não estão balanceados. A execução destas funções no Google Colab demorou cerca de uma hora e meia, mas executando a mesma ferramenta em uma IDE local, o Jupyter Notebook, fez com que este tempo decaísse para cerca de quinze minutos. Ao fim deste processo, foi criado um diretório train com 80% dos dados e test com os 20% restantes.

Para acessar as imagens nos diretórios utilizou-se as classe Path() do módulo python pathlib. Instanciando esta classe, é possível acessar as imagens através dos caminhos de diretórios de cada imagem. O comando

```
1 path_base = Path('/content/drive/MyDrive/Colab Notebooks/dados/train')
```

instancia a classe Path() para acessar os dados de treinamento que estão na pasta train.

Com os comandos

```
Eighth_Note = (path_base/'Eighth-Note').ls().sorted()
Sharp = (path_base/'Sharp').ls().sorted()
Time2_4 = (path_base/'2-4-Time').ls().sorted()
Time3_4 = (path_base/'3-4-Time').ls().sorted()
Whole_Note = (path_base/'Whole-Note').ls().sorted()
Time4_4 = (path_base/'4-4-Time').ls().sorted()
Natural = (path_base/'Natural').ls().sorted()
Flat = (path_base/'Flat').ls().sorted()
Barline = (path_base/'Barline').ls().sorted()
C_Clef = (path_base/'C-Clef').ls().sorted()
Common_Time = (path_base/'Common-Time').ls().sorted()
Cut_Time = (path_base/'Cut-Time').ls().sorted()
Dot = (path_base/'Dot').ls().sorted()
Double_Sharp = (path_base/'Double-Sharp').ls().sorted()
F_Clef = (path_base/'F-Clef').ls().sorted()
G_Clef = (path_base/'G-Clef').ls().sorted()
```

foram acessados os dezesseis diretórios correspondentes aos símbolos selecionados neste trabalho e foi criada uma lista com todas as imagens contidas neles. Para tanto, foi utilizado o método ls(), implementado para objetos da classe Path, que pertence à biblioteca fastai.

Para verificar e visualizar as imagens de qualquer uma das 16 listas criadas é possível usar a classe open() do módulo Image da biblioteca PIL (Python Imaging Library). Com os comandos

```
im_F_Clef = F_Clef[8]
im = Image.open(im_F_Clef)
im
```

obtém-se a visualização de uma imagem de treinamento do símbolo musical Clave de Fá:

Através de uma análise exploratória dos dados que compõem as imagens, observou-se que elas são compostas por três canais, ou seja, três sub-imagens sobrepostas, resultando na imagem original. Entretanto, observou-se que os três

canais apresentam a mesma informação. Portanto, optou-se por realizar um fatiamento do array, removendo dois canais. Isto permite uma economia na entrada de dados na rede neural.

Outro procedimento realizado na fase de preparação dos dados foi a normalização dos valores que os pixels podem apresentar, visto que os pixels mais escuros possível têm valor máximo de 255. Este procedimento é recomendado visto que a rede neural propaga esses valores por meio da multiplicação de parâmetros, o que pode resultar na propagação de erro numérico dentro do modelo. Por fim, os dados de cada imagem de treinamento foram armazenados em array do tipo tensor, que é próprio do pacote pytorch, o qual foi usado para implementar a rede neural convolucional. Com os comandos

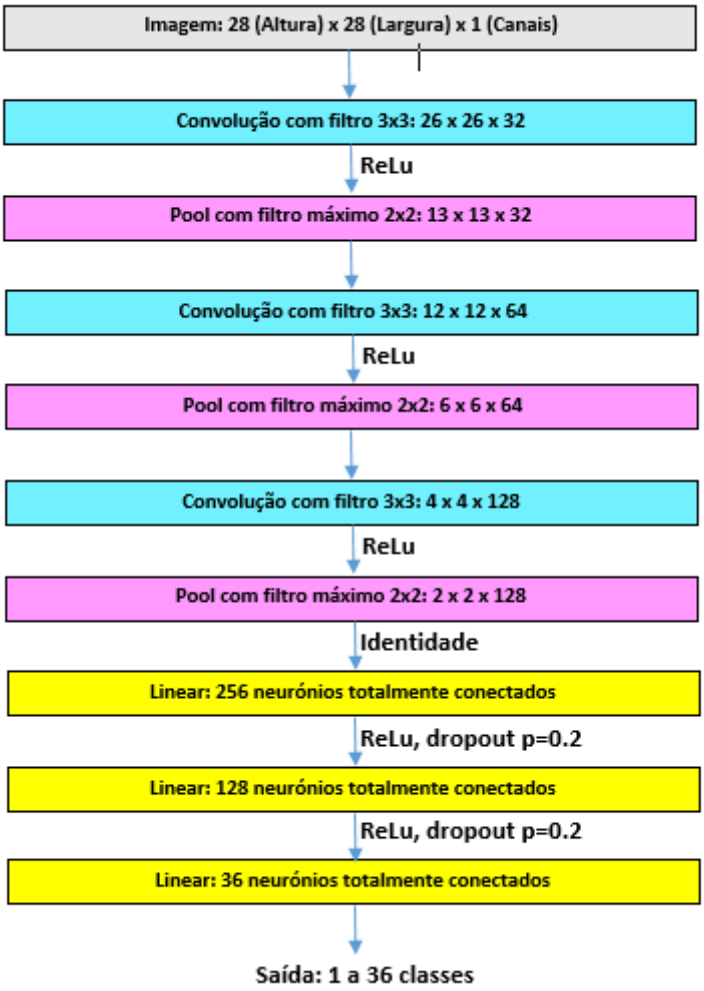
```
Eighth_Note_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Eighth_Note]
Sharp_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Sharp]
Time2_4_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Time2_4]
Time3_4_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Time3_4]
Whole_Note_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Whole_Note]
Time4_4_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Time4_4]
Natural_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Natural]
Flat_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Flat]
Barline_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Barline]
C_Clef_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in C_Clef]
Common_Time_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Common_Time]
Cut_Time_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Cut_Time]
Dot_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Dot]
Double_Sharp_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in Double_Sharp]
F_Clef_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in F_Clef]
G_Clef_tensors = [(tensor(invert(Image.open(o)))[:,:,0]).reshape(192,96) for o in G_Clef]
```

foram criadas as listas de tensores das imagens de treinamento.

3.2 Criação do Modelo de ML

Para criar a rede neural convolucional foram utilizados recursos e funcionalidades da biblioteca pytorch. A figura 5 mostra um diagrama representando a arquitetura da CNN proposta neste projeto. Esta rede recebe a imagem como dado de entrada armazenado em um tensor que apresenta formato 192 por 96 pixels.

Figura 5 – Diagrama mostrando a arquitetura da rede convolucional utilizada neste trabalho. Foram utilizadas três camadas convolucionais, todas ativadas com a função ReLu, seguidas do procedimento MaxPooling. Ao final da rede, foram utilizadas duas camadas completamente conectadas, finalizando com a camada de saída com ativação sofmax.



Fonte: Elaborada pelos autores. Link para o código: https://colab.research.google.com/drive/1ARVdH2yjod0a48av4te7PsucxX79Hanx?usp=sharing.

3.3 Treinamento da Rede Neural

No caso de uma rede neural, o aprendizado se dá à medida em que seus parâmetros são atualizados com o aumento de acertos na predição das imagens. Este processo de aprendizagem é iterativo, ou seja, a rede vai aprimorando

constantemente sua capacidade preditiva através de uma sucessão de ajustes (updates) dos parâmetros.

O processo de update dos parâmetros começa por meio de uma avaliação do erro de predição da rede neural, a partir dos dados de treinamento. Esta avaliação do erro é feita utilizando uma Função de Custo. Existem várias Funções de Custo que podem ser utilizadas. Optamos por utilizar a Entropia Cruzada (*Cross Entropy*), que é uma das Funções de Custo mais indicadas para problemas de classificação.

A partir da Função de Custo é possível determinar o ajuste dos parâmetros calculando o gradiente daquela função de maneira que se busque o melhor ajuste para minimizar o valor da Função de Custo. No caso de uma rede neural, o cálculo do gradiente é feito pelo algoritmo Backpropagation. Todo este procedimento foi realizado com as funcionalidades encontradas na biblioteca pytorch. Todos os detalhes da implementação podem ser encontrados neste link https://colab.research.google.com/drive/1ARVdH2yjod0a48av4te7PsucxX79Hanx?usp=sharing.

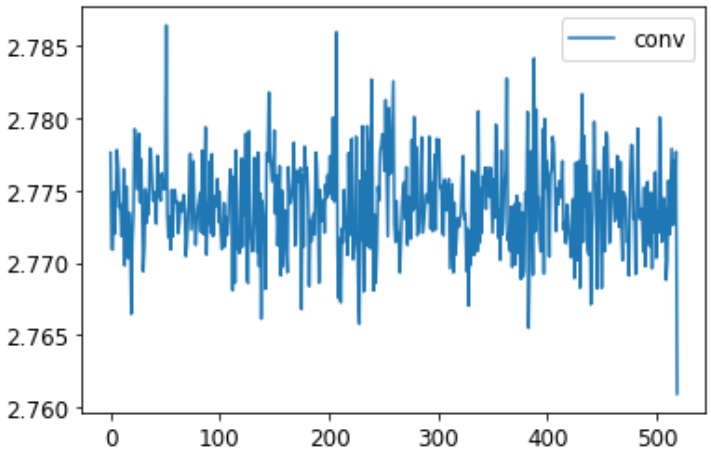
A última fase, a fase de teste, permite avaliar o modelo com dados não utilizados no treinamento para verificar se a rede neural realmente foi capaz de aprender e não apenas memorizar as respostas anteriores (nesta etapa não ocorre a atualização dos pesos, apenas verifica se eles estão bem calibrados). A ideia é que a fase de teste seja uma representação de como o algoritmo pode performar em um contexto real, no qual surgem imagens nunca vistas antes pelo algoritmo.

4 RESULTADOS E DISCUSSÃO

Ao utilizar a implementação da arquitetura proposta neste trabalho e outras mais simples com menos parâmetros, foi possível observar que após a fase de treinamento os modelos de CNN não foram capazes de aprender.

A figura 6 mostra a evolução da função de custo ao longo das atualizações dos pesos e bias. Como é possível notar, o valor da função de custo flutuou em torno de um valor constante, o que mostra que o erro pode ser reduzido ao longo da fase de aprendizagem. Este gráfico é um indicativo de que a rede não aprendeu a classificar as imagens de símbolos musicais.

Figura 6 – O gráfico mostra a evolução da entropia cruzada entre as saídas esperadas e preditas pelo modelo de CNN criado neste trabalho. A flutuação em torno de um valor constante está indicando que o modelo não conseguiu aprender a classificar as imagens.



Fonte: Elaborada pelos autores.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foram estudadas as redes neurais convolucionais considerando aspectos teóricos e práticos. A implementação e uso destas redes foi alcançado usando o pacote pytorch.

Ao implementar a CNN, deparou-se com o problema do não aprendizado da rede, em outras palavras, a rede neural implementada passou pela fase de treinamento sem ser capaz de classificar as imagens de forma assertiva. Acredita-se que a impossibilidade do aprendizado seja atribuída ao tamanho do banco de dados utilizado que apresenta uma quantidade de exemplos relativamente baixa.

Como perspectiva futura, deve-se testar a implementação proposta neste trabalho em banco de dados maiores. Por exemplo, pode-se testar a implementação usando os dados do dataset de imagens de dígitos MNIST, que possui, aproximadamente dez mil imagens para cada dígito, em contraste com HOMUS que possui trezentas imagens de cada símbolo.

Caso seja confirmado que o problema do não-aprendizado seja a quantidade baixa de dados, pode-se recorrer a redes pré-treinadas. Neste caso, é possível realizar o aprendizado mesmo com poucos dados de treinamento, visto que as camadas convolucionais de tais redes pré-treinadas têm a capacidade de reconhecer

grande variedade de formas de uma imagem, bastando apenas recompor as últimas camadas para o dataset específico e realizar apenas o treinamento delas.

REFERÊNCIAS

CS231n. Convolutional Neural Networks. Disponível em: https://cs231n.github.io/convolutional-networks/. Acesso em 3 dez. 2021.

HOMUS. The Handwritten Online Music Symbols (HOMUS) dataset. Disponível em: https://grfia.dlsi.ua.es/homus/. Acesso em 3 dez. 2021.

NIELSEN, M. Neural networks and deep learning. Disponível em: http://neuralnetworksanddeeplearning.com. Acesso em: 3 dez. 2021.

PYTORCH. From Research to Production. Disponível em: https://pytorch.org/. Acesso em: 3 dez. 2021.