

Pokeduelo

F. Erialdo D. Freitas



SAULO wants
to fight!

Modelo

- Foi implementada uma arquitetura de ***Convolutional Neural Network*** para codificação das imagens dos pokemons para o duelo (***CNNEncoder***).
- Também foi implementada uma arquitetura de ***Multilayer Perceptron*** (***DuelClassifier***), para classificar o resultado dos duelos, a partir dos *embeddings* gerados pela *CNNEncoder*.

Modelo (cont.)

Principais características da arquitetura:

- **CNNEncoder:**

- Entrada: sprites dos Pokémon com 4 canais (ex.: RGB + canal extra).
- Três blocos convolucionais com Batch Normalization, ReLU e MaxPooling:
 - Conv(32 filtros, 3x3) → MaxPool (96×96 → 48×48)
 - Conv(64 filtros, 3x3) → MaxPool (48×48 → 24×24)
 - Conv(128 filtros, 3x3) → MaxPool (24×24 → 12×12)
- Saída: vetor de embedding de 128 dimensões, obtido via camada totalmente conectada.

Modelo (cont.)

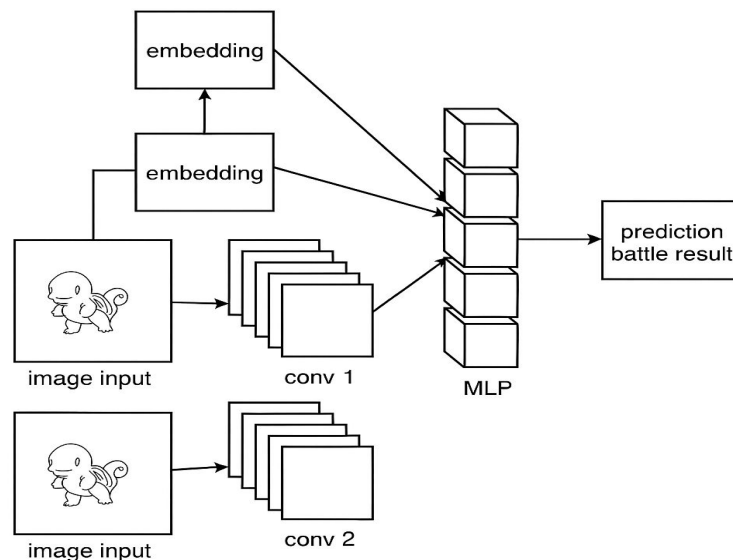
Principais características da arquitetura:

- **DuelClassifier (MLP):**
 - Recebe dois embeddings (um para cada Pokémon).
 - Combina os embeddings: $[\text{embed1}, \text{embed2}, |\text{embed1} - \text{embed2}|]$, resultando em um vetor de $3 \times 128 = 384$ dimensões.
 - Estrutura MLP:
 - Linear($384 \rightarrow 256$) + ReLU + Dropout(0.5)
 - Linear($256 \rightarrow 128$) + ReLU + Dropout(0.5)
 - Linear($128 \rightarrow 3$) (saída com logits para classificação: vitória, derrota ou empate).

Modelo (cont.)

- Imagem que ilustra a arquitetura montada no experimento:

- **Parte 1 (*CNNEncoder*):** Rede convolucional que extrai embeddings compactos das imagens dos Pokémon.
- **Parte 2 (*MLP DuelClassifier*):** Combina os embeddings dos dois Pokémon e produz a predição do resultado do duelo.

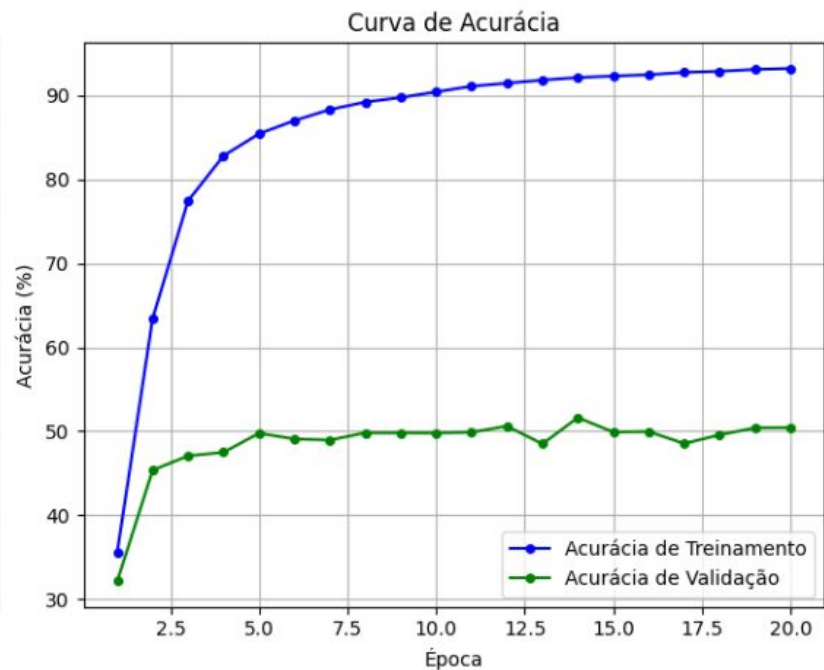
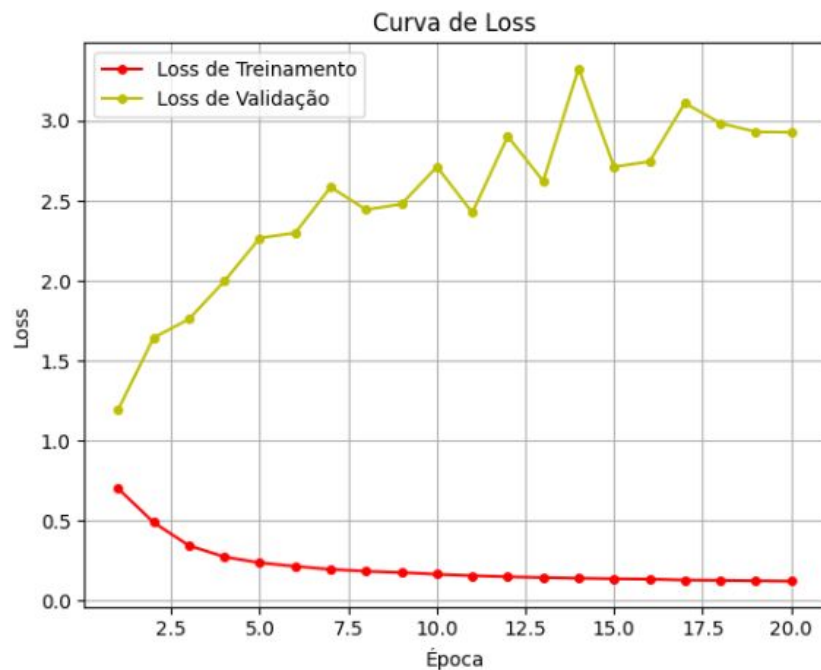


Modelo (cont.)

- **Processo de treinamento e definição dos hiperparâmetros:**
 - O treinamento do modelo se deu de forma até tranquila, com boas taxas de acurácia e baixo loss, mesmo que essas tenham iniciado baixas, ao decorrer das épocas (escolhi **20 épocas**);
 - **Função de perda (loss):** CrossEntropyLoss (como a saída da rede são logits);
 - **Otimizador:** Adam;
 - **Treinamento padrão** em *minibatches*, com tamanho de batch igual a 32;
 - **Regularização** via *Dropout* na MLP para evitar *overfitting*;
 - Também foram implementadas técnicas como **Weighted Cross-Entropy Loss** para o *criterion* da Função de Perda e Regularização **L2** com *weight_decay* igual a $1e-4$.

Treinamento

- Plots das taxas loss e acurácia durante o treinamento:



Treinamento (cont.)

- Report de Classificação & Matriz de Confusão

--- Relatório de Classificação no Conjunto de Teste ---

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

Derrota	0.64	0.57	0.61	9279
---------	------	------	------	------

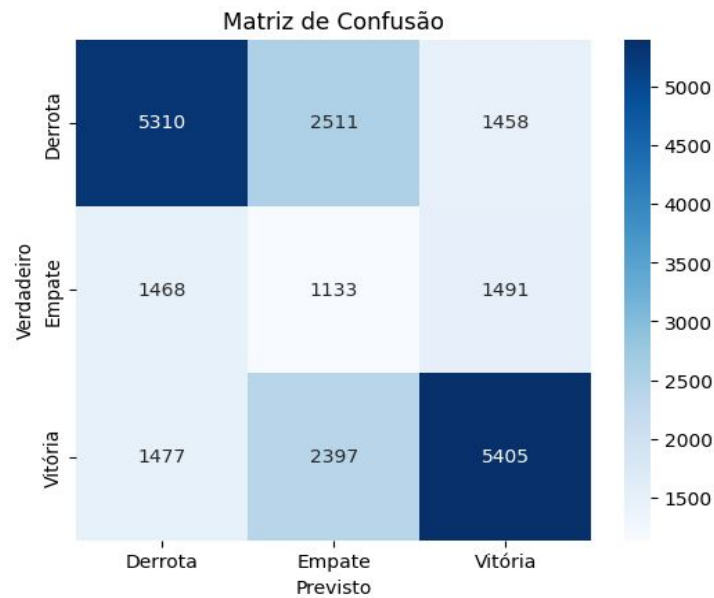
Empate	0.19	0.28	0.22	4092
--------	------	------	------	------

Vitória	0.65	0.58	0.61	9279
---------	------	------	------	------

accuracy			0.52	22650
----------	--	--	------	-------

macro avg	0.49	0.48	0.48	22650
-----------	------	------	------	-------

weighted avg	0.56	0.52	0.54	22650
--------------	------	------	------	-------



Avaliação subjetiva

- Pontuação de **coerência semântica**: 7/15 (46,67%).

--- Avaliação de Coerência Semântica (Casos Específicos) ---

Caso 1: Pikachu vs Gyarados

- Expectativa Lógica: Pikachu vence
- Previsão do Modelo: Derrota (Label: 0)
- Resultado: ✖ Incorreto

Caso 2: Charizard vs Blastoise

- Expectativa Lógica: Charizard perde
- Previsão do Modelo: Vitória (Label: 2)
- Resultado: ✖ Incorreto

Caso 3: Venusaur vs Charizard

- Expectativa Lógica: Venusaur perde
- Previsão do Modelo: Derrota (Label: 0)
- Resultado: ✔ Correto

Caso 4: Machop vs Alakazam


- Expectativa Lógica: Machop perde
- Previsão do Modelo: Vitória (Label: 2)
- Resultado: ✖ Incorreto

Caso 5: Golem vs Blastoise


- Expectativa Lógica: Golem perde
 - Previsão do Modelo: Vitória (Label: 2)
 - Resultado: ✖ Incorreto
-

Avaliação subjetiva (cont.)


Caso 6: Scyther vs Charizard

- Expectativa Lógica: Scyther perde
- Previsão do Modelo: Derrota (Label: 0)
- Resultado:  Correto


Caso 7: Jigglypuff vs Gengar

- Expectativa Lógica: Jigglypuff perde
- Previsão do Modelo: Derrota (Label: 0)
- Resultado:  Correto


Caso 8: Electabuzz vs Rhydon

- Expectativa Lógica: Electabuzz perde (sem efeito)
- Previsão do Modelo: Empate (Label: 1)
- Resultado:  Incorreto


Caso 9: Magikarp vs Machop

- Expectativa Lógica: Magikarp perde
- Previsão do Modelo: Derrota (Label: 0)
- Resultado:  Correto


Caso 10: Caterpie vs Arcanine

- Expectativa Lógica: Caterpie perde
 - Previsão do Modelo: Derrota (Label: 0)
 - Resultado:  Correto
-


Caso 11: Farfetch'd vs Zapdos

- Expectativa Lógica: Farfetch'd perde
- Previsão do Modelo: Derrota (Label: 0)
- Resultado:  Correto


Caso 12: Ditto vs Dragonite

- Expectativa Lógica: Ditto perde (na prática)
- Previsão do Modelo: Empate (Label: 1)
- Resultado:  Incorreto


Caso 13: Metapod vs Charizard

- Expectativa Lógica: Metapod perde
- Previsão do Modelo: Derrota (Label: 0)
- Resultado:  Correto

Caso 14: Snorlax vs Hitmonlee

- Expectativa Lógica: Snorlax perde
- Previsão do Modelo: Vitória (Label: 2)
- Resultado:  Incorreto

Caso 15: Dragonite vs Articuno

- Expectativa Lógica: Dragonite perde
 - Previsão do Modelo: Empate (Label: 1)
 - Resultado:  Incorreto
-

Relate principais achados

Foram encontradas algumas questões, que trouxeram certa dificuldade para treinar o modelo, principalmente na parte da classificação. Também encontrei um problema ao baixar um dos *sprites*. Basicamente, os problemas encontrados foram:

- Problema ao baixar o *sprite* id 678, onde não consegui identificar o motivo;
- Treinamento um pouco lento;
- De início identifiquei que o modelo não estava generalizando bem, pois observei as métricas (acurácia e *loss*, no modelo como todo) bem baixas no conjunto de validação, mesmo com boas taxas no conjunto de treinamento, indicando possível *overfitting*.
 - Observei que o modelo estava evoluindo bem no conjunto de treinamento, mas no conjunto de validação de início tinha taxas baixíssimas de acurácia (entre ~20% e ~25%), e taxas de *loss* um pouco altas (entre ~1.8 e ~2.5). Então fiz os ajustes e implementações de técnicas como *Batch Normalization*, na rede CNN, e de ajustes em alguns hiperparâmetros, como diminuição no *Learning Rate*, implementação de ponderações para as classes no *criterion* da função de perda (implementando *Weighted Cross-Entropy Loss*), adicionando regularização L2 (passando *Weight Decay* = $1e-5$ e depois $1e-4$, no otimizador *Adam*), onde obtive uma melhora significativa, observei que a acurácia de validação subia, mas o *loss* também subia razoavelmente na mesma proporção;
 - Uma questão identificada foi que o modelo erra mais nas previsões da Classe “Empate”, talvez por essa ser uma classe que tem muito menos registros, em relação às outras duas; sendo um dos motivos pelo qual, depois de algumas pesquisas, implementei o *Weighted Cross-Entropy Loss*;