Deep Learning - Homework 2

Brian Nunes - 1105399
Mariana Serrão - 1105045

10/01/2022

1 Questão 1

1.1 Exercício 1

1.1.1 Item (a)

Dada uma imagem N×N×D, que passa por uma *layer* com filtro F×F×D, K canais e *stride* S, a dimensão de seu resultado será MxMxK, com M dado por:

$$M = (N - F)/S + 1 \tag{1}$$

Dessa forma, ao passar pela layer convolucional, a dimensão da imagem será dada por:

$$H' = (H - M)/1 + 1 = H - M + 1 \tag{2}$$

$$W' = (W - N)/1 + 1 = W - N + 1 \tag{3}$$

Portanto, a dimensão de z é (H-M+1) x (W-N+1) x 1

1.1.2 Item (b)

Podemos representar um elemento de $z' = Mx'$ como:

$$z'(i) = \sum_{j=1}^{HW} M(i,j) \cdot x'(j) \tag{4}$$

Dessa forma, é possível perceber que cada elemento M(i,j) denota a parcela de x'(j) em z'(i). Ou seja, M é a matriz composta pelos pesos W(a,b), que multiplicados por x'(j) contibuem para z'(i). Assim, concluímos que o elemento M(i,j) pode ser descrito como:

$$\begin{aligned} a = (j - (i-1) \cdot S)//W + 1 \\ b = (j - (i-1) \cdot S)\%W + 1 \\ 0 < a \wedge b \leq F \Rightarrow M(i,j) = W(a,b) \\ \text{Caso contrário, } M(i,j) = 0 \end{aligned} \tag{5}$$

1.1.3 Item (c)

Dada uma imagem com 3 canais e um unico filtro MxN, a quantidade de parâmetros da rede é $3 \times M \times N$. Já para uma *fully connected layer*, a quantidade de parâmetros é dada pela multiplicação da dimensão do input (imagem) pela do output (h2): $HW \times 3 \times (H - M + 1) \times (W - N + 1)/2$.

1.2 Exercício 2

As probabilidades do *Attention* são dadas pelo produto escalar das matrizes de *query*, chave e valor com a entrada x0. Como as matrizes de projeção são 1x1, o produto escalar se reduce ao produto elemento a elemento. Então, as probabilidades de atenção são dadas por:

$$(x_0 * x_0) \tag{6}$$

E o *output* do *Attention* é dada por:

$$(x_0 * (x_0 * x_0)) \tag{7}$$

2 Questão 2

2.1 Exercício 1

Uma rede *fully-connected* tem mais parâmetros que uma CNN, pois todos os outputs dependem de todos os inputs, ou seja, todos os neurônios estão conectados com todos os inputs, assim a quantidade de parâmetros livres em uma *layer* é dada pelo produto da dimensão dos inputs pelos outputs. Enquanto isso, em uma CNN, filtros percorrem toda a extensão do input, reduzindo a quantidade dos parâmetros livres para o produto da dimensão dos filtros (FxFxD) pela quantidade de filtros da *layers*.

2.2 Exercício 2

Algumas das razões pelas quais CNNs tendem a obter melhores resultados na generalização de padrões são:

- As redes neurais convolucionais performam bem no reconhecimento de padrões espaciais, utilizando múltiplas layers que aprendem diferentes níveis de abstrações. Assim, as primeiras camadas aprendem os padrões mais simples (e.g. curvas) e as últimas aprendem os mais complexos (e.g. objetos).
- Em CNNs é comum o uso de *pooling layers*, que atribuem invariância translacional, rotacional e escalar. Ou seja, a rede lida melhor com as variações espacieais, conseguindo detectar padrões mesmo que estejam em posições diferentes.
- A quantidade de parâmetros previamente mencionada também ajuda a evitar o overfitting.

2.3 Exercício 3

Nesse caso, é provável que a CNN não obtenha um resultado melhor do que a rede *fully-connected*. Como não haveria uma estrutura espacial, a CNN não se beneficiaria da invariancia ou do reconhecimento hierárquico de padrões, podendo não encontrar *features* úteis. Enquanto isso, a *fully-connected network* é mais flexível, computando melhor relações entre features sem conexão espacial.

2.4 Exercício 4

A melhor configuração foi utilizando um *learning rate* igual a 0.0005, como pode ser visto na tabela abaixo.

Learning Rate	0.01	0.0005	0.00001
Final Trainig Loss	0.6347	0.0439	0.3727
Final Validation Accuracy	0.8144	0.9845	0.9562
Final Test Accuracy	0.6791	0.9577	0.8942

Table 1: Performance por *Learning Rate*

Adicionalmente foram gerados gráficos de *training loss* e *validation accuracy* em função do *epoch*.

2.5 Exercício 5

Observando os *Activation Maps* gerados pelas redes é possível perceber um destaque para diferentes características da imagem. Inicialmente são destacados pequenos padrões, como cantos e pequenas curvas. Em seguida, são identificados traçados e curvas maiores, seguidos de uniões de traços. Finalmente são identificados múltiplos traçados e curvas formando padrões mais complexos, se assemelhando mais a imagem original.

3 Questão 3

3.1 Exercício 1

3.1.1 Item (a)

3.1.2 Item (b)

3.1.3 Item (c)

Uma maneira de melhorar os resultados sem mudar a arquitetura do modelo seria usar a busca por feixe durante o processo de decodificação na função *test()*. Isso pode levar a previsões mais precisas e diversas

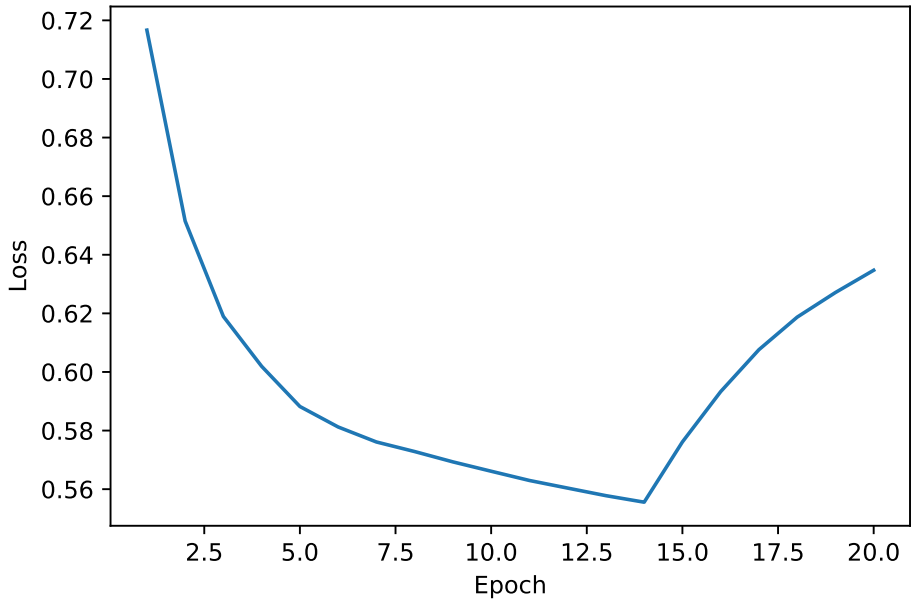

Figure 1: *Trainig Loss* por *epoch* com *Learning Rate* 0.01

evitando o problema de o modelo ficar preso em uma solução localmente ótima mas subótima durante a decodificação. Podemos implementar mudando a operação *argmax* usada no *loop* de decodificação para um algoritmo de busca por feixe.

4 Responsabilidades

O projeto foi produzido em conjunto, sendo divididos igualmente os exercícios entre os membros do grupo. Nomeadamente, Mariana foi responsável pela segunda questão, enquanto Brian pela terceira. Já a primeira questão, foi dividida, sendo o primeiro exercício realizado por Mariana, e o segundo por Brian. No que se refere ao relatório, Mariana foi responsável pela estruturação do documento em LaTex, e ambos foram responsáveis pelo preenchimento dos exercícios que resolveram.

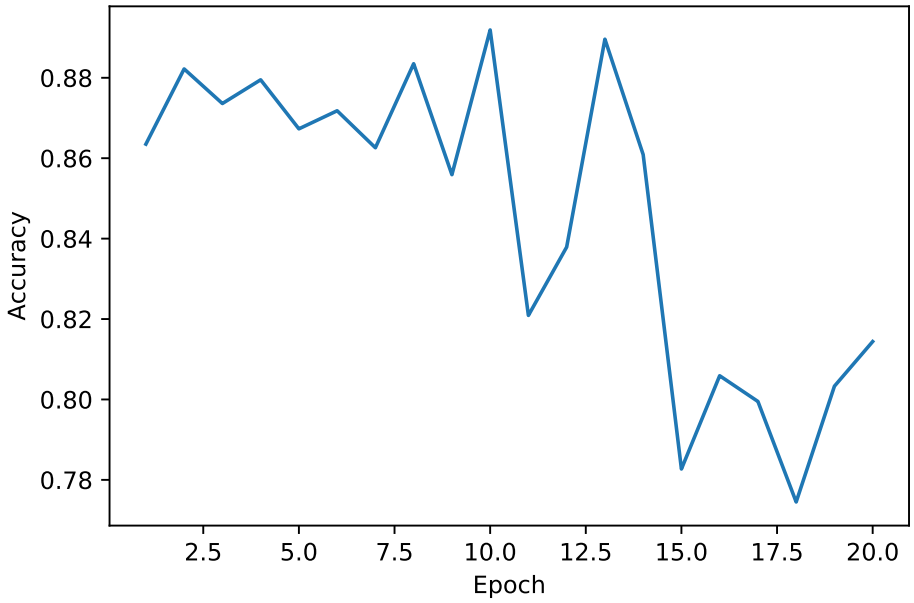

Figure 2: *Validation Acuracy* por *epoch* com *Learning Rate* 0.01

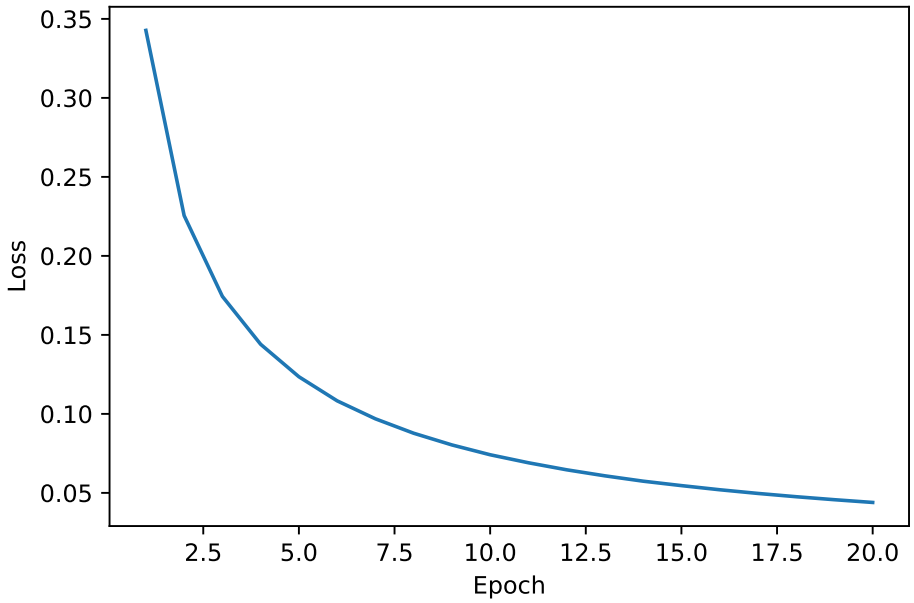

Figure 3: *Training Loss* por *epoch* com *Learning Rate* 0.0005

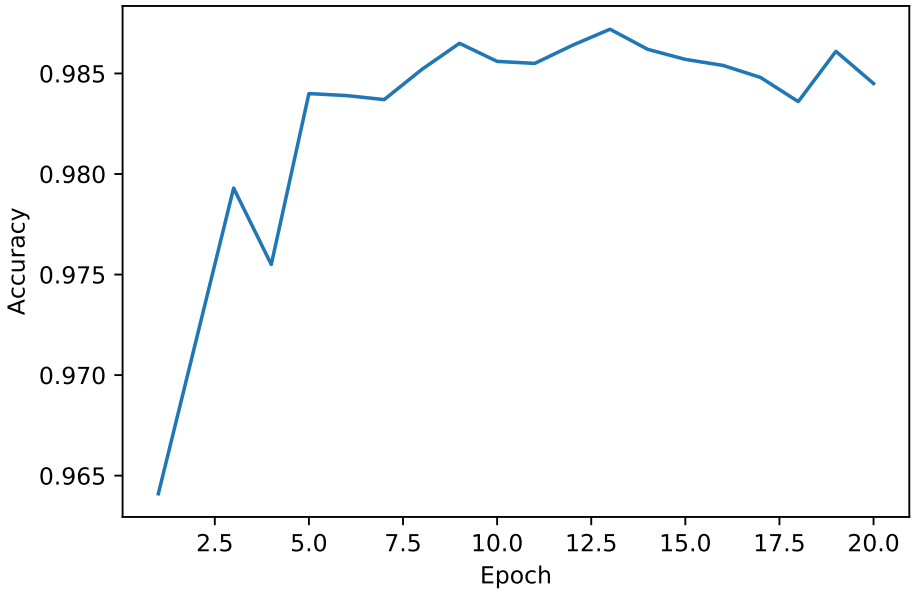

Figure 4: *Validation Acuracy* por *epoch* com *Learning Rate* 0.0005

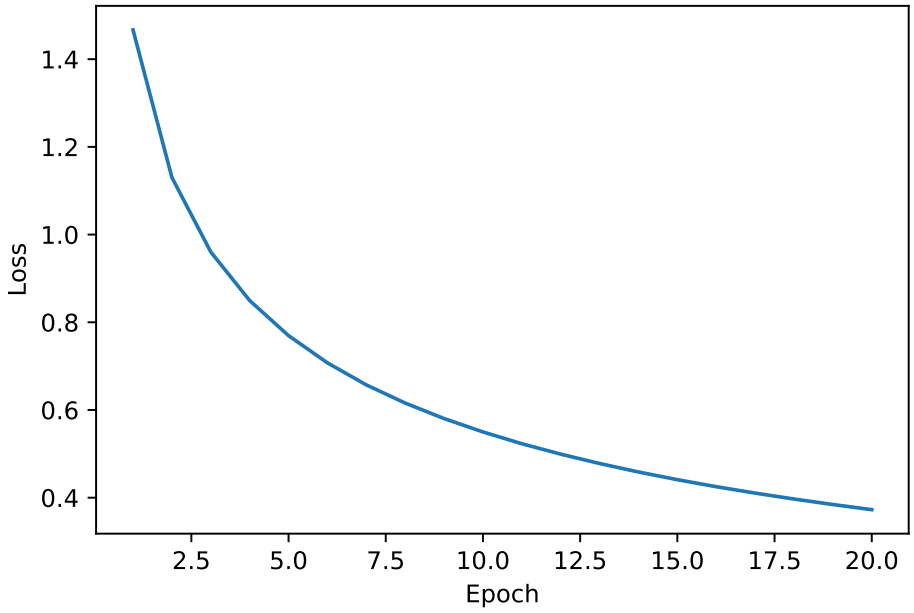

Figure 5: *Training Loss* por *epoch* com *Learning Rate* 0.00001

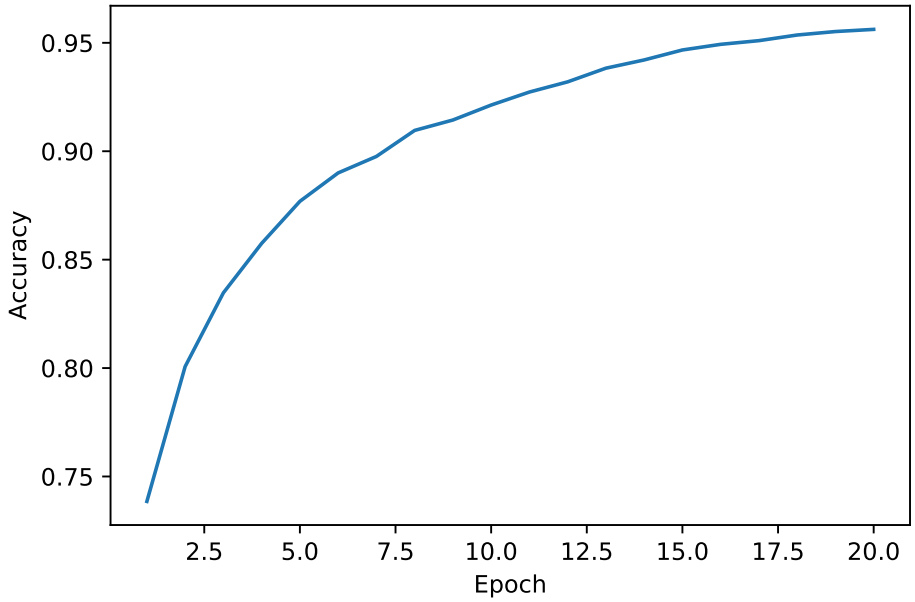

Figure 6: *Validation Acuracy* por *epoch* com *Learning Rate* 0.00001

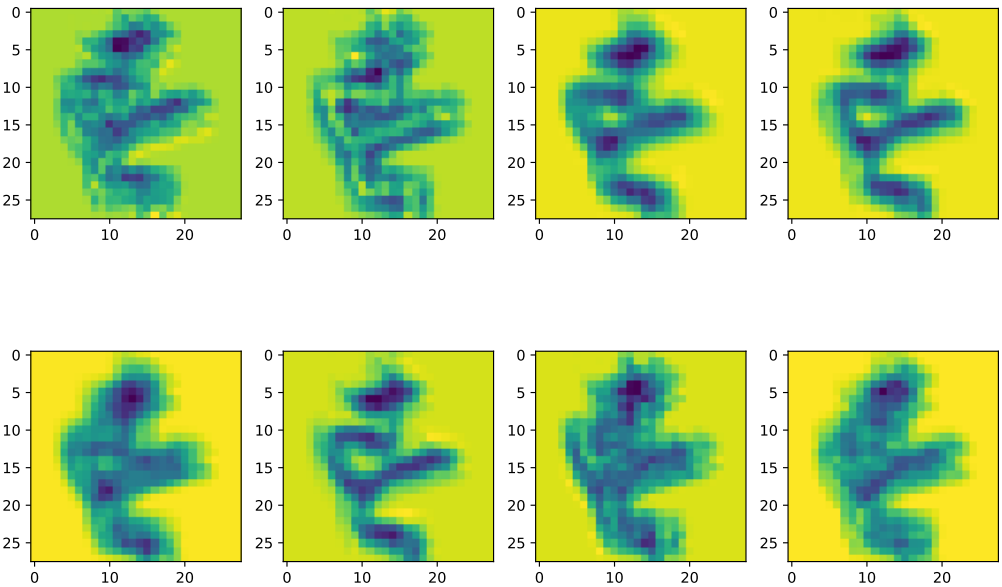

Figure 7: *Activation Map* com *Learning Rate* 0.01

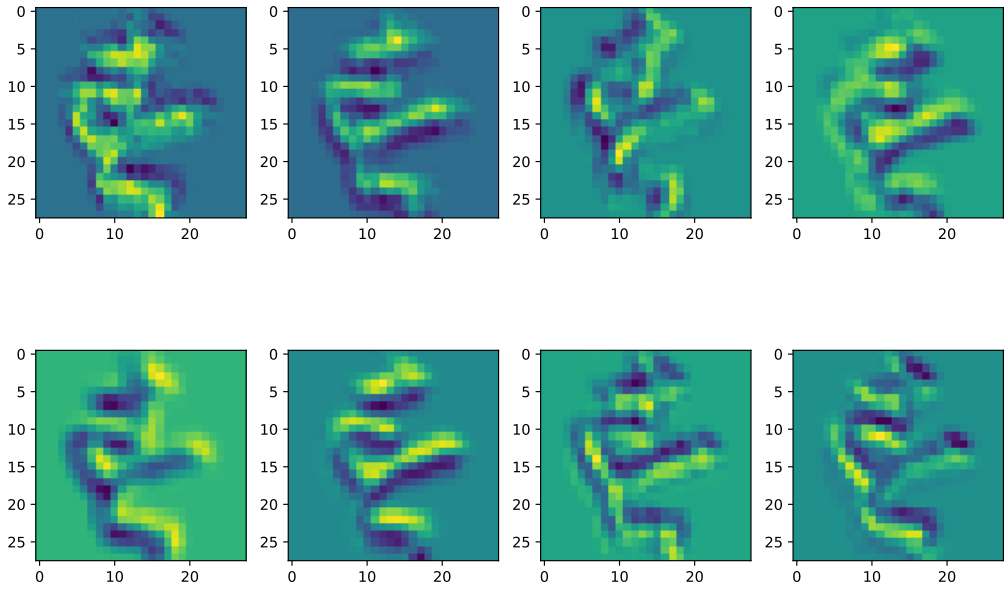

Figure 8: *Activation Map* com *Learning Rate* 0.0005

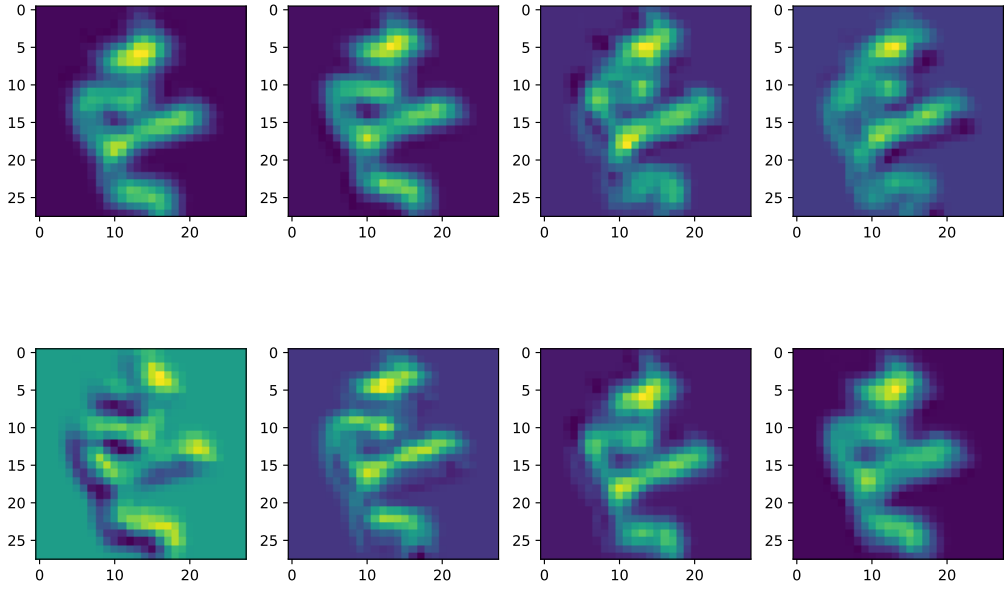

Figure 9: *Activation Map* com *Learning Rate* 0.00001

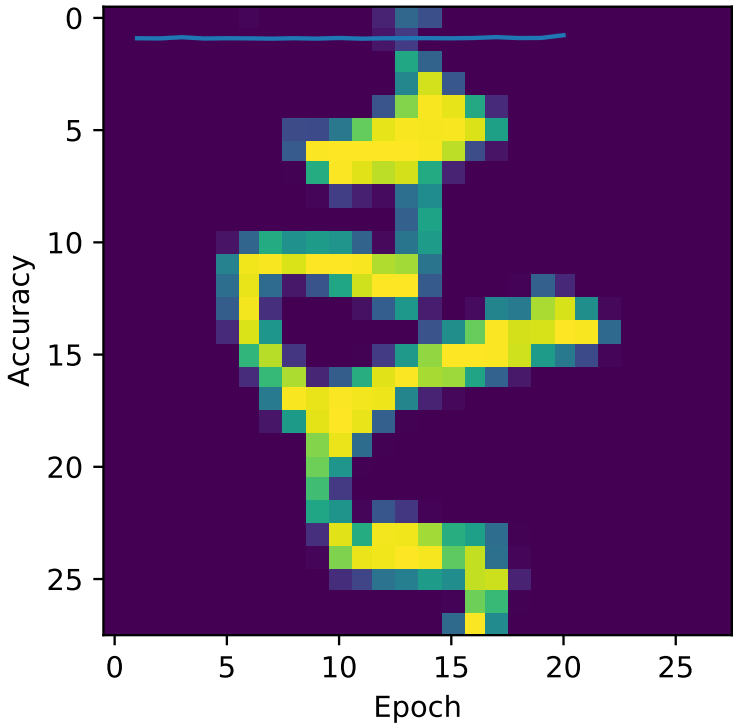

Figure 10: Imagem original

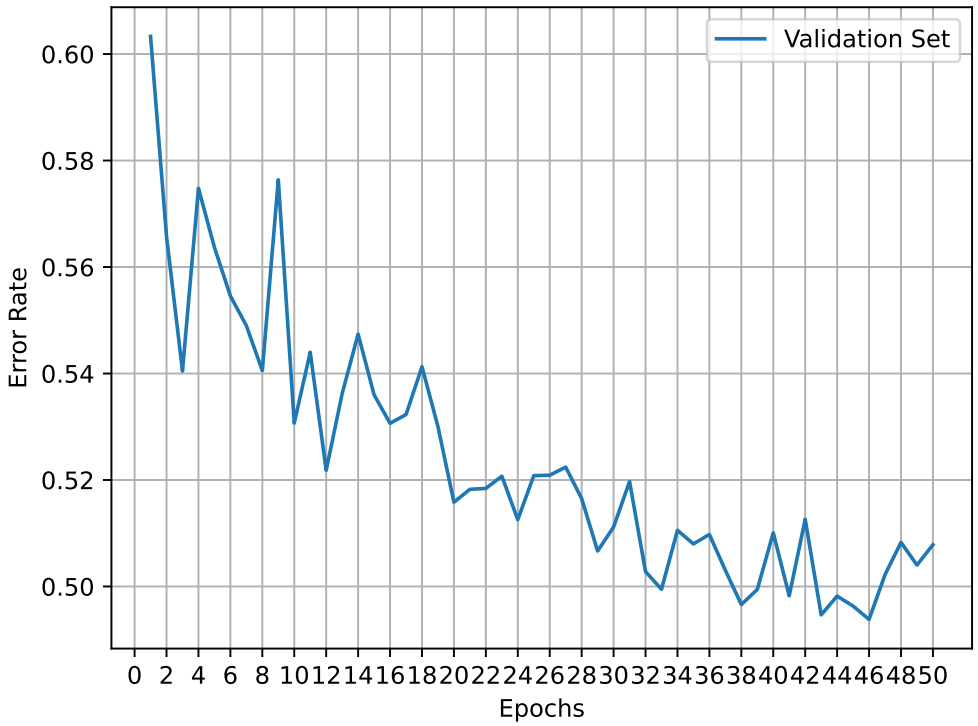

Figure 11: Taxa de erro sem utilização de *Attention*

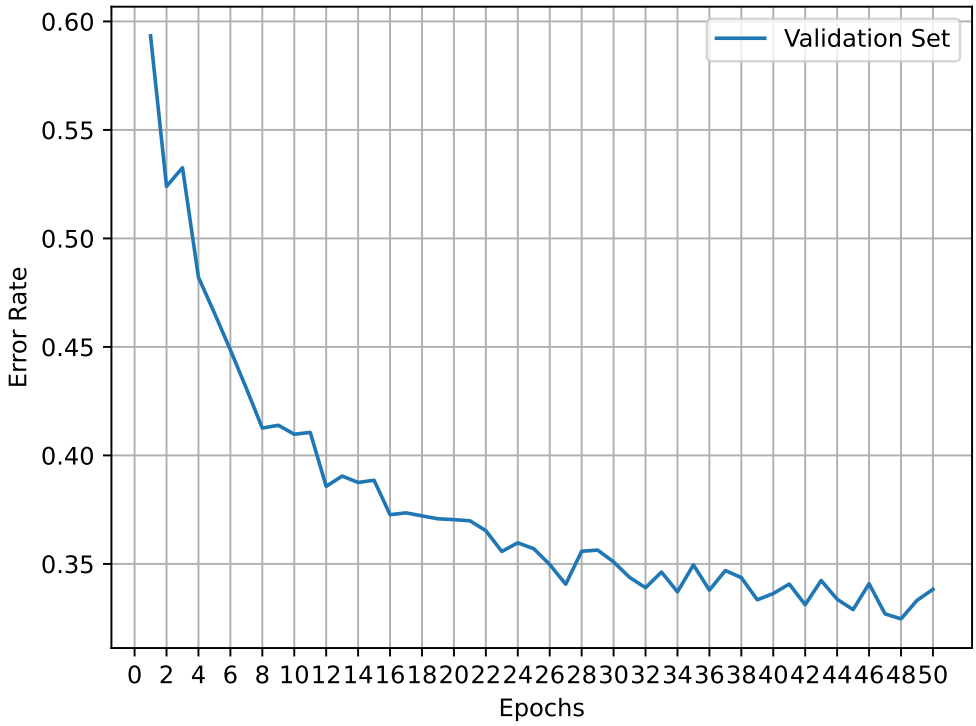

Figure 12: Taxa de erro com utilização de *Attention*