Exercicio Programa 1

Pedro Semcovici (12745511)

[1]Universidade de São Paulo (EACH-USP)
Av Arlindo Bettio 1000, São Paulo, Brazil

pedrosemcovici@usp.br

Resumo. *Nesse trabalho foi explorado a comparação entre modelo de redes neurais fully connected (FCNN) e redes neurais convolucionais (CNN) para a classificação de imagens do conjunto Kuzushiji-MNIST, que é um conjunto de imagens de 10 possíveis caracteres do Hiragana. Além do experimento básico da comparação entre os dois algoritmos, foi realizado testes removendo a camada de polling da CNN e, também, realizando uma etapa de data augmentation em ambos os algoritmos aumentando o conjunto de treino em 10 vezes.*

1. Introdução

Este trabalho trata-se de um exercício programa (EP) da disciplina MAC5921 - Deep Learning do programa de pós-graduação em Ciência da Computação do IME-USP. A proposta desse trabalho consiste em comparar o desempenho de CNN (Convolucional Neural Network) e FCNN (Fully Connected Neural Network) para a classificação de imagens. Ao longo desse trabalho, serão respondidas algumas das perguntas levantadas no enunciado do trabalho, sendo elas:

- Q1 - Como a performance de uma rede neural totalmente conectada se compara com a de uma CNN ao classificar imagens de um dataset específico?
- Q2 - Qual é a distribuição das classes no dataset que você escolheu? Existem classes desbalanceadas?
- Q3 - Quais são as principais diferenças entre uma rede neural totalmente conectada (fully connected) e uma rede neural convolucional (CNN) em termos de arquitetura?
- Q4 - Como as operações de convolução em uma CNN ajudam na extração de características de uma imagem?
- Q5 - Qual é o papel das camadas de pooling em uma CNN? Seria possível treinar uma CNN sem elas? O que aconteceria?
- Q6 - Como você pode visualizar e interpretar os filtros (kernels) aprendidos por uma CNN?
- Q7 - O que essas visualizações dizem sobre o tipo de características que a CNN aprende nas primeiras camadas versus nas últimas?
- Q8 - É possível interpretar os pesos de uma rede totalmente conectada da mesma forma? Por que ou por que não?

2. Materiais e Métodos

2.1. Conjunto de dados

O conjunto de dados utilizado para este trabalho é o Kuzushiji-MNIST [Clanuwat et al. 2018] disponivel em um repositório do GitHub[1]. Nesse repositório há a

[1]https://github.com/rois-codh/kmnist

possibilidade de download de 3 datasets (Kuzushiji-MNIST, Kuzushiji-49 e Kuzushiji-Kanji), o escolhido para esse trabalho foi o Kuzushiji-MNIST dado ser o conjunto com a menor quantidade de classes, o que facilita os experimentos.

O Kuzushiji-MNIST possui um total de 10 classes, sendo cada uma delas um caracter do Hiragana. Uma representação de cada uma das classes pode ser vista abaixo, sendo cada "linha" uma classe, a primeira "coluna" sendo um bom exemplo (digital) do caracter e as outras sendo exemplos escritos à mão.

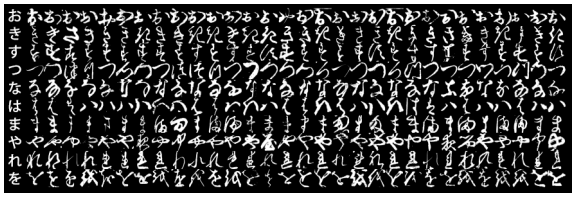

Figure 1. Exemplos de imagens do Kuzushiji-MNIST

O conjunto de dados já possui uma separação de treino e teste, sendo 60 mil imagens no conjunto de treino e 10 mil no de teste. Além disso, o conjunto é perfeitamente balanceado, possuindo exatamente 10% das observações para cada uma das classes, sendo assim 6 mil observações para cada classe no treino e mil para o teste.

As imagens do cojunto de dados são de baixa resolução, tendo dimensões (28,28,1). As imagens são disponibilizadas em formato numpy, o que facilita a leitura dos dados.

2.2. Experimentos realizados

Os modelos treinados são CNN e FCNN, além de testar algumas alterações para ver se essas melhoram ou pioram o desempenho. Assim resultando nos seguintes experimentos:

1. **CNN**: modelo CNN simples
2. **FCNN**: modelo FCNN simples
3. **CNN com data augmentation**: modelo CNN com dados aumentados em 10x
4. **FCNN com data augmentation**: modelo FNN com dados aumentados em 10x
5. **CNN sem camada de polling**: modelo CNN removendo a camada de polling

Os modelos CNN e FCNN seguem as estruturas demonstradas na Figuras 2 e 3, respectivamente. Tendo apenas a mudança de retirar a camada de max_polling2d no caso do experimento "CNN sem camadda de polling".

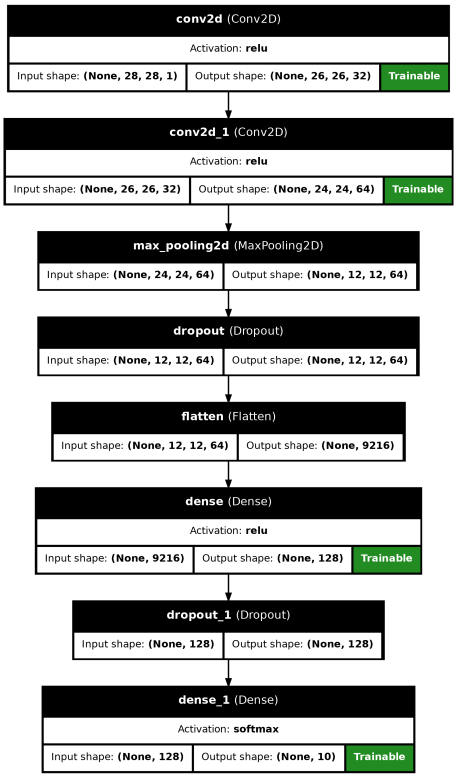

Figure 2. Model CNN

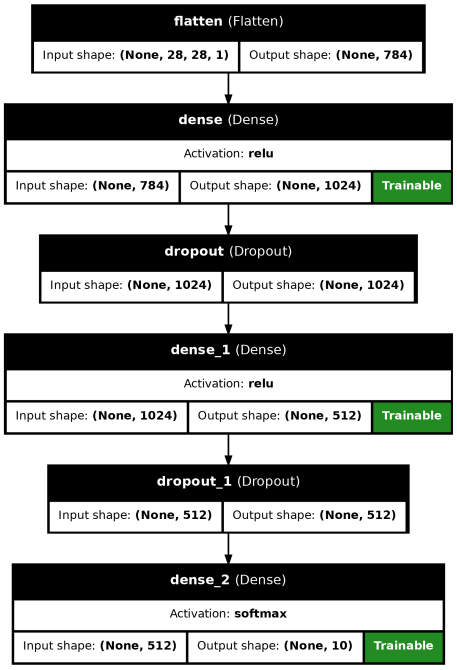

Figure 3. Modelo FCNN

Uma das exigências no enunciado é que o modelo FCNN e CNN tenham uma quantidade similar de parâmetros treinaveis. Assim sendo, o modelo FCNN (Figura 3) possui 1333770 parâmtros treináveis, enquanto o modelo de CNN (Figura 2) possui 1199882 parâmetros treináveis.

Para realizar o treinamento, predição e outras tarefas relacionadas aos modelos anteriores, foi utilizado o tensorflow [Abadi et al. 2015] e keras [Chollet et al. 2015].

Abaixo segue algumas configurações feitas nos modelos, fora as demostradas nas Figuras 2 e 3:

1. **Early stopping:**Os modelos não possuem um número fixo de épocas para o treinamento, foi estabelecido um número arbitráriamente grande de 10 mil épocas e, quando o valor de loss na validação não diminui em 10 épocas seguidas o treinamento é parado e o melhor modelo escolhido é aquele que possui o menor valor de loss na validação.
2. **Reduce learning rate on plateu:** A taxa de aprendizado dos modelos inicia como 0.001, porém, quando o valor de loss na validação começa a ter poucas variações, essa taxa de aprendizado é reduzida. Isso faz com que os pesos mudem menos ao longo das épocas quando o treinamento está chegando em um mínimo, assim trazendo uma mais precisão.
3. **Função de loss:** A função de loss escolhida foi a *categorical crossentropy*, pois dentro as funções de loss implementadas no keras é a que mais atendia as especificações do projeto.
4. **Otimizador:** O otimizador escolhido foi o adadelta

2.3. Execução do código

O código utilizado para gerar os resultados apresentados nesse relatório se encontra em um repositório do GitHub [2] e as instruções para a execução se encontram no arquivo README.md na seção "Instruções".

3. Resultados

Na tabela 1, temos os resultados de *F1-Score* para cada uma das 10 classes no conjunto de teste, bem como o resultado de *F1-Score macro* (média aritimética do *F1-Score* das classes), que nos dá uma visão geral do desempenho do modelo para todas as classes.

classe	CNN	CNN com data augmentation	CNN sem polling	FCNN	FCNN com data augmentation
0	0.9114	0.9508	0.9168	0.9127	0.9514
1	0.8728	0.9042	0.8867	0.8778	0.9107
2	0.8483	0.8850	0.8354	0.8409	0.8676
3	0.9223	0.9259	0.9202	0.9260	0.9187
4	0.8681	0.9034	0.8737	0.8739	0.9112
5	0.9013	0.9263	0.8976	0.9074	0.9232
6	0.8765	0.9272	0.8906	0.8890	0.8996
7	0.9038	0.9444	0.9025	0.9008	0.9471
8	0.8694	0.9170	0.8773	0.8763	0.9211
9	0.9002	0.9388	0.9016	0.8994	0.9409
macro avg	0.8874	0.9223	0.8902	0.8904	0.9192

Table 1. Resultados de *F1-Score* no conjunto de teste (arredondados em 4 casas decimais)

É possível ver que, em todos os modelos, o desempenho em cada classe é bastante consistente, não tenho classes que possuem uma grande vantagem em detrimento das outras e, também, não há classes em grande desvantagem.

Considerando o *F1-Score macro*, vemos que os diferentes experimentos não causaram grandes diferenças nos resultados, tendo o pior resultado sendo a CNN com 88.74% de *F1-Score macro* e o melhor sendo a CNN com data augmentation com 92.23% de *F1-Score macro*.

Remover a camada de polling traz ganhos aparentemente não significativos ao desempenho do modelo no conjunto de teste, tendo em vista que há uma diferença de aproximadamente 0.26% nos resultados. Já a adição de *data augmentation* se mostra eficiente na melhora do desempenho nos dois algoritmos testados, FCNN e CNN.

Por outro lado, quando olhamos a curva de aprendizado da CNN (Figura 4) e da FCNN (Figura 5), vemos que a CNN converge por volta de 600 épocas, enquanto a FCNN precisa de quase o dobro de épocas para convergir.

[2] https://github.com/semcovici/kmnist-classification

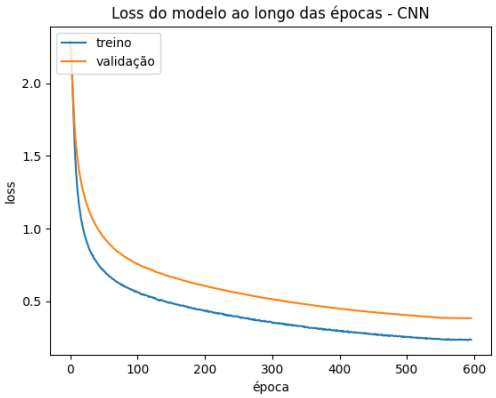

Figure 4. Curva de aprendizado CNN

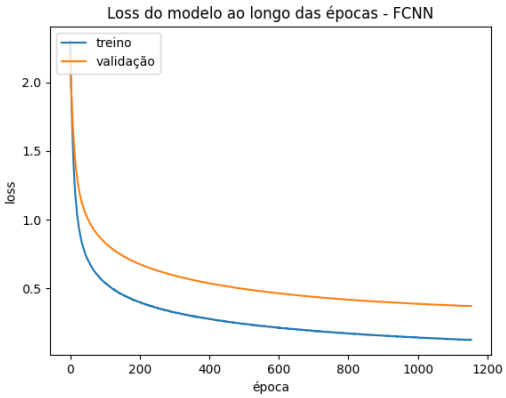

Figure 5. Curva de aprendizado FCNN

Já a remoção da camada de polling da CNN faz o modelo convergir mais rápido, porém vemos um *overfitting* do modelo no treino, tendo um loss menor que 0.25 no conjunto de treino no fim do treinamento, que é menor que o loss final do conjunto de treino na CNN tradicional, como demostrado na Figura 6.

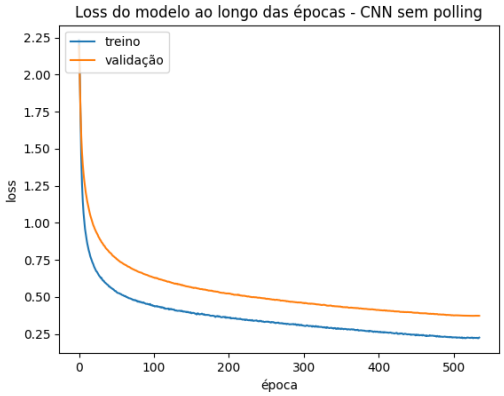

Figure 6. Curva de aprendizado CNN sem polling

A adição do *data augmentation* na CNN traz uma redução drástica do *overfitting* do modelo no treinamento, tendo o loss ao longo das épocas no treino e validação bem próximos, conforme demostrado na Figura 7. Já na FCNN a adição da etapa de data augmentation também traz uma redução no *overfitting*, porém não tanto quando a CNN, conforme demostrado na Figura 8. Além disso, a adição de *data augmentation* mostra uma redução no número de épocas para convergir. Em contraponto a isso, a adição de mais dados faz com que seja mais custoso computacionamente treinar o modelo, de forma que a redução de épocas não diminui o tempo de treinamento.

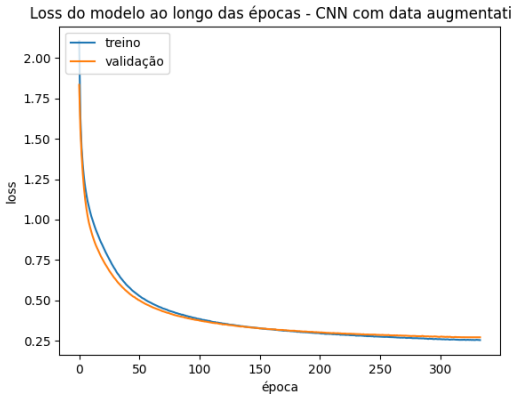

Figure 7. Curva de aprendizado CNN com *data augmentation*

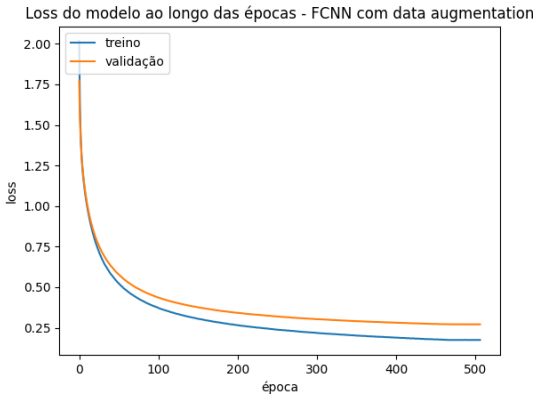

Figure 8. Curva de aprendizado FCNN com *data augmentation*

Nas Figuras 9 e 10, temos os filtros formados após o treinamento do modelo CNN nas camadas convolucionais. Não há algum padrão interpretável nesses filtros que nos mostre algum aspecto que o modelo aprendeu nas imagens. Isso não significa que o modelo não aprendeu nada nas imagens, isso apenas significa que a análise dos filtros gerados não resulta em nenhum aspecto de melhor entendimento do funcionamento do modelo.

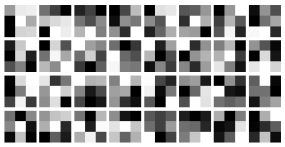

Figure 9. Filtros da primeira camada convolucional do modelo de CNN

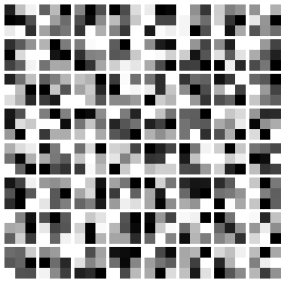

Figure 10. Filtros da segunda camada convolucional do modelo de CNN

Esse padrão de "filtros não interpretáveis" se repete na versão sem camada de polling e na versão com *data augmentation*.

4. Conclusão e limitações

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Chollet, F. et al. (2015). Keras.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. (2018). Deep learning for classical japanese literature. *CoRR*, abs/1812.01718.