Impactos de *Data Augmentation* em *Convolutional Neural Network* (CNN)

Thiago Prado de Campos[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
ACF Centro Politécnico – Jd. Das Américas – CEP 81531-980 – Curitiba – PR

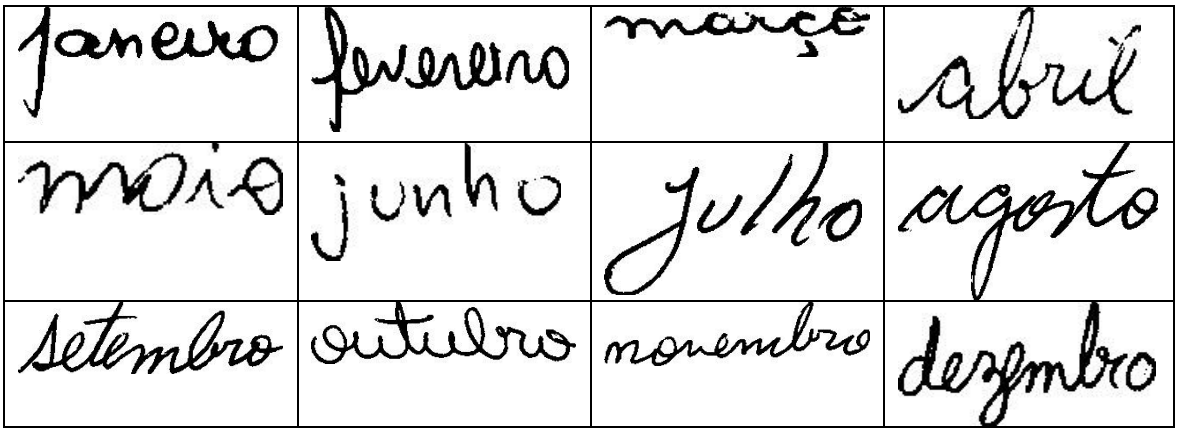
contato@thiagotpc.com

Resumo. *Este é um relatório de atividade de laboratório da disciplina de Aprendizagem de Máquina (2020/Período Especial).*

1. Objetivo

Nesta atividade o objetivo é classificar imagens que contém texto manuscrito que representam meses do ano no idioma português (Quadro 1 - Exemplo de Imagens da Base de Treinamento e Testes). A classificação deve ser feita por meio de *Convolutional Neural Network (CNN)*. Particularmente, a LeNet5 [LeCun 2020] e um outro modelo de de CNN qualquer a ser escolhido. O treinamento das redes deve ocorrer com os dados originalmente fornecidos e com dados de treinamento aumentados artificialmente (*data augmentation*), comparando os resultados de ambas situações.

Quadro 1 - Exemplo de Imagens da Base de Treinamento e Testes Original



A implementação das redes foi feita com a linguagem Python e a biblioteca Keras [Keras SIG 2020a]. A execução de todos os scripts aqui apresentados aconteceu em *notebook* de uso doméstico equipado com processador Intel i7, 16GB RAM e placa de vídeo Nvidia Geforce 830M.

2. Usando LeNet5 Sem *Data Augmentation*

Considerando a modelagem em Keras da LeNet5 apresentada em aula, realizamos testes variando os seguintes parâmetros de entrada: dimensão do vetor de características (considerando 32x32, 64x64, 60x17 e 120x34) e; número de épocas (10, 20, 50).

A dimensão foi variada para verificar se um maior número de características permitia melhor resultado e se o mesmo poderia acontecer considerando a proporção de maior largura que altura das imagens, característica do texto escrito e observada pela

base originalmente fornecida. Os valores propostos para as dimensões retangulares consideram a proporção média de aproximadamente 3.5 entre largura/altura encontrada nas imagens, ao mesmo tempo que a quantidade de características ficam próximas das dimensões quadradas (Tabela 1).

Tabela 1 - Dimensões e Quantidade de Características

Dimensão	Quantidade de Características
32 x 32	1024
64 x 64	4096
60 x 17	1020
120 x 34	4080

Os resultados obtidos da execução dos treinamentos e testes para cada um dos parâmetros foram registrados. A Tabela 2 traz, para cada cenário, a acurácia da validação. Em arquivos de saída salvos durante a execução é possível consultar informações mais completas, como perdas, matriz de confusão e outras.

Tabela 2 - Acurácia por Dimensão e Épocas de Treinamento

Dimensão / Épocas	32x32	64x64	60x17	120x34
	formato quadrado		largura maior que altura	
10	0.42394	0.63591	0.51371	0.58603
20	0.59600	0.68079	0.56608	0.65087
40	0.67581	**0.69326**	0.68329	0.68827

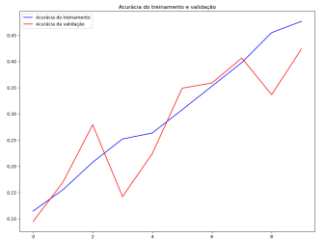
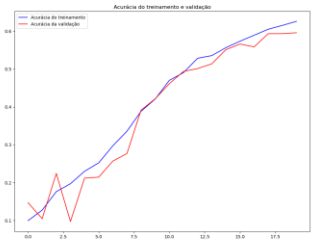
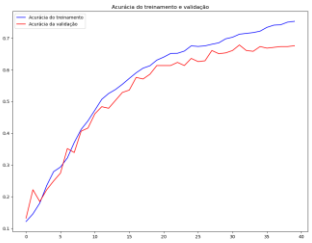
Os resultados apresentados de acurácia, sugerem que:

A) Dimensões com maior largura que altura não trouxeram necessariamente melhores resultados. Comparando a mesma quantidade de características, as dimensões de maior largura que coluna foram piores em 4 configurações e melhores em 2 configurações;

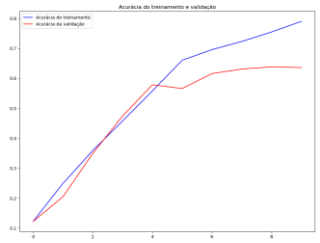
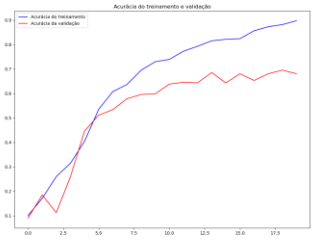
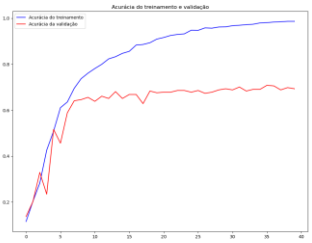
B) Com aproximadamente 4 mil características (64x64 e 120x34), o ganho de acurácia é menor ao dobrar o número de épocas de 20 para 40.

Os quadros abaixo (1, 2, 3 e 4) mostram a acurácia de treinamento (azul) e validação (vermelho) à medida que se aumentou o número de épocas, para cada dimensão de característica utilizada. Percebe-se que com menos características, a curva de acurácia na validação acompanhou melhor a curva de acurácia do treinamento. A distância maior entre as linhas a partir da 20ª época nas dimensões 64x64 e 120x34 refletem o que foi afirmado no item B acima.

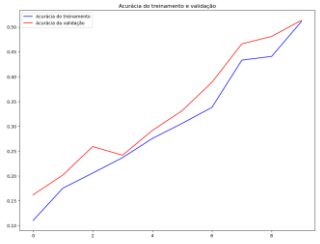
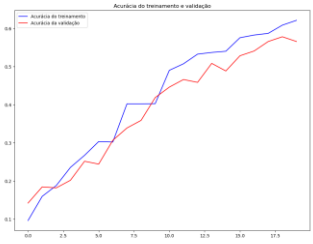
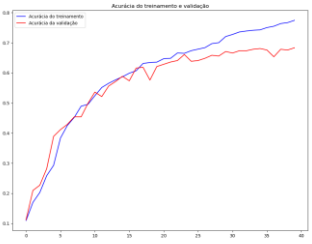
Quadro 2 - Curva de acurácia em relação a épocas na dimensão 32x32

10	20	40
		

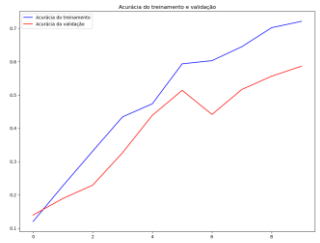
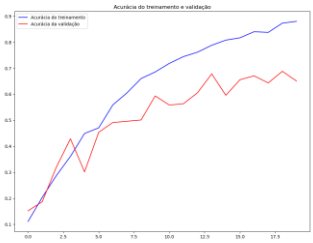
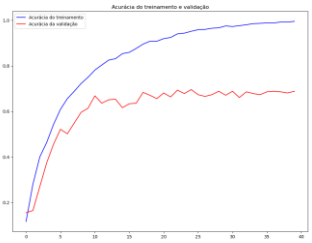
Quadro 3 - Curva de acurácia em relação a épocas na dimensão 64x64

10	20	40
		

Quadro 4 - Curva de acurácia em relação a épocas na dimensão 60x17

10	20	40
		

Quadro 5 - Curva de acurácia em relação a épocas na dimensão 120x34

10	20	40
		

2. Usando outra CNN, sem *Data Augmentation*

Depois de realizar os testes usando a LeNet5, foi modelada uma outra rede CNN (Quadro 6). O tempo de execução deste modelo foi sensivelmente maior.

Quadro 6 - Código para outro modelo de CNN usada

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(8, 8), activation='relu',
input_shape=(img_rows, img_cols, 3)))
model.add(Conv2D(128, (2, 2), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))
```

Começamos a sequência com duas convoluções, que funcionam como filtros para capturar traços marcantes. Inicialmente com 32 filtros de tamanho 8x8 e depois 128 filtros de tamanho 2x2. A terceira camada é um *pooling* para simplificar a informação da camada anterior. Usamos o método de *maxpooling* padrão para diminuir a quantidade de pesos e evitar *overfitting*. Na quarta camada temos um *Dropout* para "desligar" alguns neurônios ocultos da rede, com objetivo de "forçar a rede a aprender a classificar por caminhos diferentes". Depois vem as camadas de *Flatten*, para sair das duas dimensões para um vetor, *Dense* com a função de ativação *relu*, mais um *Dropout* e a última camada *Dense*, com uma função de ativação *softmax* para a quantidade de classes que desejamos classificar.

Para estes modelos, executamos o treinamento e validação para os mesmos parâmetros de dimensões e épocas usadas na LeNet5 e apresentamos os resultados de acurácia na Tabela 3.

Tabela 3 - Acurácia por Dimensão e Épocas de Treinamento para Outra CNN

Dimensão / Épocas	32x32	64x64	60x17	120x34
10	0.32418	0.48379	0.34413	0.39401
20	0.56857	0.72817	0.52867	0.69576
40	0.74064	**0.81795**	0.70324	0.78553

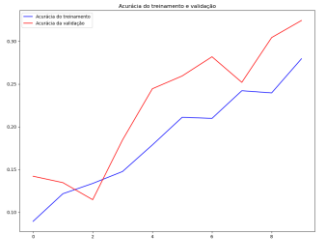
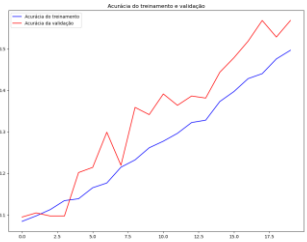
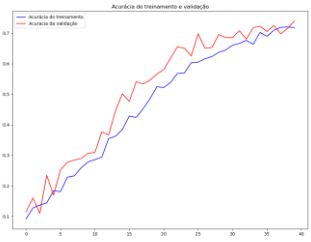
Os resultados apresentados sugerem que:

A) Dimensões com maior largura que altura não trouxeram melhores resultados. Comparando a mesma quantidade de características e épocas, as dimensões de maior largura que coluna foram piores em 5 configurações e melhor em apenas uma configuração;

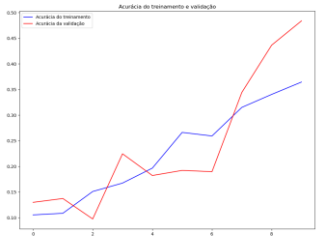
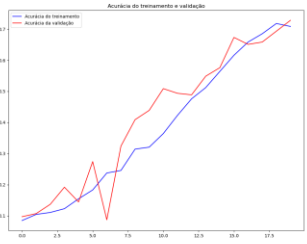
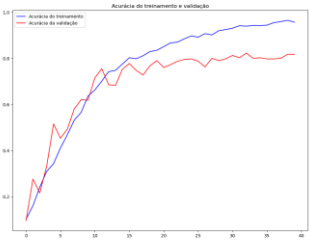
B) Neste modelo de CNN, o aumento das épocas contribuiu sensivelmente para melhor da acurácia. Com 40 épocas, a acurácia superou a LeNet5 em todas as dimensões.

Os quadros abaixo (1, 2, 3 e 4) mostram a acurácia de treinamento (azul) e validação (vermelho) à medida que se aumentou o número de épocas, para cada dimensão de característica utilizada. Diferente do comportamento da LeNet5, nesta CNN, houve maior volatilidade da curva de acurácia na validação em relação a de treinamento. Quanto maior o número de épocas, menor a variação.

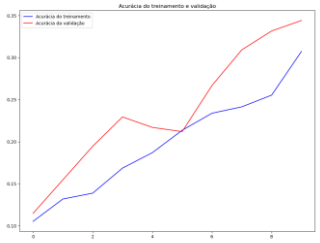
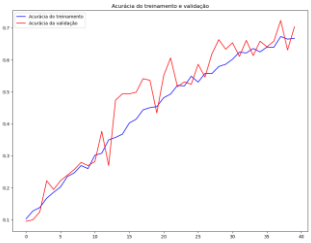
Quadro 7 - Curva de acurácia em relação a épocas na dimensão 32x32

10	20	40
		

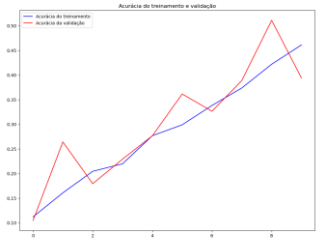
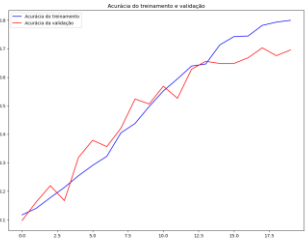
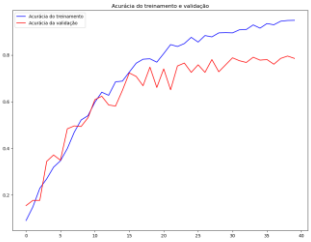
Quadro 8 - Curva de acurácia em relação a épocas na dimensão 64x64

10	20	40
		

Quadro 9 - Curva de acurácia em relação a épocas na dimensão 60x17

10	20	40
		

Quadro 10 - Curva de acurácia em relação a épocas na dimensão 120x34

10	20	40
		

4. Estratégias de *Data Augmentation*

A base de dados de treinamento fornecida originalmente para o experimento continha 1578 imagens. Com objetivo de obter melhor treinamento, o passo seguinte do experimento procurou aumentar a base de dados de treinamento, utilizando três estratégias: (1) gerar imagens de texto automaticamente com uso de fontes manuscritas; (2) modificar as imagens de treinamento originalmente fornecidas por meio de uma biblioteca especializada em modificação de texto em imagens para reconhecimento de caracteres [Luo et al. 2020] e; (3) por meio de modificação das imagens originalmente fornecidas aplicando-se funções de: redimensionamento desproporcional e leve rotação.

Não foram aplicadas modificações por meio de *flip* e *shift* pois entende-se que tais mudanças podem ao invés de facilitar, dificultar o reconhecimento de caracteres, gerando grandes anomalias que não corresponderia ao texto esperado. Não seria natural encontrar texto manuscrito espelhado, por exemplo.

Procurou-se, para cada método, gerar uma quantidade próxima de imagens a fim de que a quantidade de imagens geradas não fosse, necessariamente, um fator discriminante ao se comparar as diferentes estratégias.

4.1. Imagens Geradas por Fontes Manuscritas

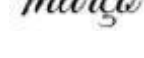
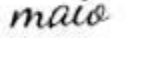
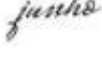
Por meio do site Google Fonts [Google LLC 2020], foram escolhidas trezes fontes da categoria *handwriting*. Então, estas foram usadas no projeto para que novas imagens fossem geradas automaticamente com ajuda da biblioteca Pyllow [Python Software Foundation 2020].

Um script Python gera, para cada mês do ano, imagens usando cada uma das fontes escolhidas e aplicando algumas variações como ângulo de rotação, tamanho de letras e tamanho da imagem. Todas usam a mesma cor de fundo (branco) e texto (preto). O ângulo de rotação varia levemente, de -9 a 9 graus, pois o objetivo aqui é simular a escrita natural, com alguma inclinação. As imagens ainda são levemente esticadas horizontal ou verticalmente. O excesso de fundo branco é cortado para ficarem semelhantes as originalmente fornecidas. O Quadro 11 mostra parte do código do *script image-generator-by-font.py* que gera as imagens por meio desta estratégia.

Quadro 11 - Trecho do código para gerar imagens baseadas em fonte

```
# gera imagens conforme parametros
def generate_months_images():
  counter = 1
  for month in months:
    for font in fonts:
      for font_size in font_sizes:
        for angle in draw_angles:
          for size in draw_sizes:
            img = Image.new('RGB', (x_size, y_syze), color=background_color)
            d = ImageDraw.Draw(img)
            font_path = fonts_dir + font + fonts_extension
            fnt = ImageFont.truetype(font_path, font_size)
            d.text((5, 5), month, font=fnt, fill=forecolor)
            filename = 'font-based-' + str(counter) + '.jpg'
            img.rotate(angle, expand=1, fillcolor=background_color).resize(
              (int(x_size*size), int(y_syze * size))).save(images_dir+'/'+ filename)
            adiciona_rotulo(month, filename)
            counter = counter + 1
    print(f'Generated {counter - 1} imagens')
```

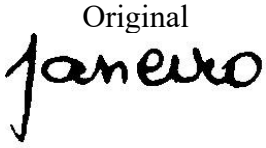
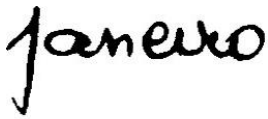
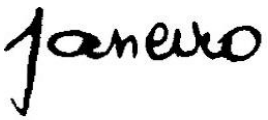
Quadro 12 - Exemplos de imagens geradas por fontes manuscritas

4.2. Imagens Geradas pela Biblioteca *Text-Image-Augmentation*

Outra estratégia para aumentar a base de treinamento foi usar uma biblioteca criada por terceiros [RubanSeven 2020] [Luo 2020] para gerar novas imagens a partir das originalmente fornecidas. Foi tomado cuidado para usar apenas as imagens separadas para treinamento, isto é, não gerar novas imagens a partir das imagens de validação. Para cada uma das 1578 imagens da base de treinamento, foram geradas 11 novas variações (Quadro 13). Esta biblioteca aplica mudanças bastante sutis na imagem original, o que pode servir de reforço para que a rede aprenda melhor cada exemplo. No projeto, o *script* responsável por gerar as imagens com base na biblioteca é o *image-generator-by-library.py*.

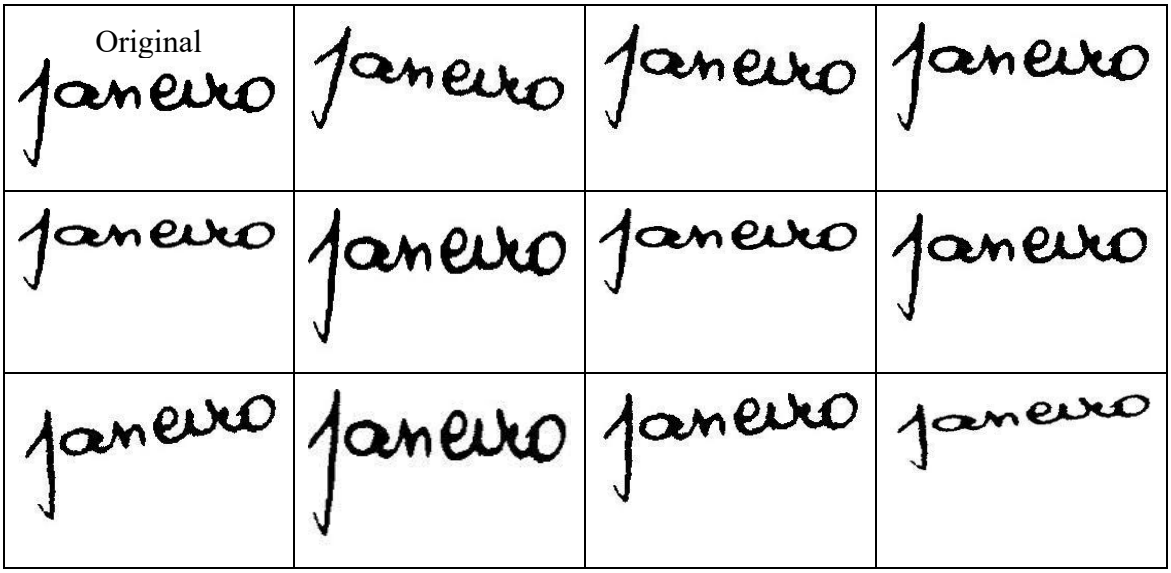
Quadro 13 - Exemplo de Imagens Geradas pela Biblioteca

Original 			
			
			

4.3. Imagens Geradas por manipulação direta das originais

A terceira estratégia para aumentar a base de treinamento foi aplicar diretamente sobre a as imagens originais, algumas funções de rotação e redimensionamento desproporcionais, usando funções da biblioteca. Para cada uma das 1578 imagens da base de treinamento, foram geradas também 11 novas variações (Quadro 13). O *script* responsável por gerara estes arquivos chama-se *image-generator-by-direct-manipulation.py*.

Quadro 14 - Exemplo de Imagens Geradas por Manipulação Direta



4.4. Nova Base de Treinamento

As imagens geradas por cada estratégia foram guardadas em uma pasta exclusiva com um arquivo de rotulagem correspondente.

Então, foi criado quatro conjuntos de base de treinamento (Tabela 4), com cada um contendo as imagens originalmente fornecidas para treinamento acrescido os novos arquivos gerados, mas separadamente, a fim de procurar entender qual teria melhor resultado. O *script* que faz a união da base original com as imagens aumentadas é o *merge-training-data.py*. As linhas das novas bases de treinamento foram embaralhadas para evitar *catastrophic forgetting*.

Em seguida, os testes descritos na seção 2 e 3 foram refeitos para 10 e 40 épocas (valores mínimos e máximos de épocas) e apenas para as dimensões 32x32 e 64x64. Esta limitação foi necessária pelo longo tempo necessário para treinamento com a base de dados aumentada.

Cada execução ainda ocorreu separadamente para cada conjunto de treinamento. O objetivo foi entender qual estratégia de *data augmentation* teria melhor resultado: a) apenas os arquivos gerados por fontes manuscritas; b) apenas os arquivos gerados usando a biblioteca de terceiro; c) apenas usando modificação própria das imagens originais ou; d) todos os três métodos juntos.

Tabela 4 – Novos Conjuntos de Treinamento

Método		**Quantidade**
1	Fonte manuscritas + originais	20.298
2	Biblioteca de terceiro + originais	18.936
3	Manipulação Direta + originais	18.936
4	Todos os métodos juntos + originais	55.014

5. Execuções com *Data Augmentation*

Primeiramente foram executados todos os cenários de base de treinamento aumentada (conjuntos 1, 2, 3 e 4) para a dimensão de 32x32. Devido ao elevado tempo de execução (treinamento + validação) de cada cenário com *data augmentation* para a dimensão de 64x64 foi realizada execução apenas para o conjunto de treinamento independente que se mostrou mais eficiente na dimensão de 32x32.

Os resultados de acurácia em cada rede para estas novas execuções são apresentadas nas seções 5.1 e 5.2, a seguir.

Tabela 5 - Tempo de Execução de Treinamento e Validação

Modelo	Conjunto de Treinamento	Dimensão de Entrada	Épocas de Treinamento	Tempo de Execução
LeNet5	1	32x32	10	374s
LeNet5	2	32x32	10	402s
LeNet5	3	32x32	10	389s
LeNet5	4	32x32	10	1056s
LeNet5	1	32x32	40	1283s
LeNet5	2	32x32	40	1192s
LeNet5	3	32x32	40	1443s
LeNet5	4	32x32	40	5093s (1h24min)
LeNet5	2	64x64	10	1801s (30min)
LeNet5	2	64x64	40	6258s (1h44min)
Outra CNN	1	32x32	10	594s
Outra CNN	2	32x32	10	527s
Outra CNN	3	32x32	10	548s
Outra CNN	4	32x32	10	1482s
Outra CNN	1	32x32	40	1912s (31min)
Outra CNN	2	32x32	40	1940s (32min)
Outra CNN	3	32x32	40	2170s (36min)
Outra CNN	4	32x32	40	6791s (1h53min)
Outra CNN	2	64x64	10	3056s (50min)
Outra CNN	2	64x64	40	*estimado 3h10min*

5.1. LeNet 5 com Data Augmentation

Percebe-se na Tabela 6 que os resultados com o conjunto de treinamento aumentado foram muito melhores do que sem. Analisando individualmente as estratégias de *data augmentation* adotadas (conjuntos 1, 2 e 3), destacam-se os resultados da abordagem

usando a biblioteca de *Text-Image-Augmentation*, maiores que os demais (entre aproximadamente 3% a 40% melhor conforme variação do número de épocas).

Outra observação é que, para dimensões de 32x32 e 40 épocas, houve pouca diferença no resultado provocado pelo uso da estratégia de imagens geradas por fontes (conjunto 1). Isto pode ser ao fato de que, o estilo de letra das fontes não ser próximo ao estilo da base original. Além disso, quanto mais treinada a rede, ela provavelmente melhor detecta o estilo da escrita existente na base original, não fazendo diferença apresentá-la a novos estilos de escritas. Por esta razão, para ganhar tempo de execução, não foi utilizado esse conjunto para dimensão de 64x64.

Com relação a modificação dos arquivos originais, a estratégia que usa a biblioteca *Text-Image-Augmentation* teve resultado melhor entre as três estratégias. Por esta razão, a execução em tamanho 64x64 com DA foi repetido apenas para este conjunto (2).

Usando *Text-Image-Augmentation*, com apenas 10 épocas e 64x64 já se obteve resultado próximo ao alcançado com 40 épocas em 32x32. Com 40 épocas e 64x64, obteve-se a melhor acurácia usando um conjunto de testes com apenas uma estratégia de *data augmentation*: 0.86034.

Tabela 6 - Resultados da LeNet5 com *Data Augmentation* sob diferentes conjuntos de treinamento

Dimensão / Épocas	32x32		64x64	
	Conjunto	**Acurácia**	**Conjunto**	**Acurácia**
10	0 *(sem DA)*	0.42394	0 *(sem DA)*	0.63591
10	1	0.54862	1	---
10	**2**	**0.76807**	**2**	**0.82294**
10	3	0.72069	3	
10	**4**	**0.81047**	4	---
40	0 *(sem DA)*	0.67581	0 *(sem DA)*	0.69326
40	1	0.68329	1	---
40	**2**	**0.82543**	**2**	**0.86034**
40	3	0.79800	3	
40	**4**	**0.87780**	4	---

5.2. Outra CNN com *Data Augmentation*

As impressões obtidas na Outra CNN com *data augmentation* foram as mesmas obtidas na LeNet5 com *data augmentation*. Com o mesmo destaque para os resultados dos conjuntos 2. Seguindo o mesmo raciocínio, para o tamanho de 64x64 foi executado apenas o experimento considerando a base aumentada por meio de *Text-Image-Augmentation* (conjunto 2).

Adicionalmente, podemos observar que o Outra CNN teve melhor desempenho que a LeNet5 sempre que se usou um conjunto com *data augmentation*. O resultado

melhor dentre os experimentos, considerando um conjunto de testes com apenas uma estratégia de *data augmentation*, aconteceu quando executada por 40 épocas na dimensão de 64x64: 0.90024 de acurácia.

Tabela 7 - Resultados da Outra CNN com *Data Augmentation* sob diferentes conjuntos de treinamento

Dimensão / Épocas	32x32		64x64	
	Conjunto	**Acurácia**	**Conjunto**	**Acurácia**
10	0 *(sem DA)*	0.32418	0 *(sem DA)*	0.48379
10	1	0.67331	1	---
10	**2**	**0.80548**	**2**	**0.86783**
10	3	0.78553	3	---
10	**4**	**0.86783**	4	---
40	0 *(sem DA)*	0.74064	0 *(sem DA)*	0.81795
40	1	0.76558	1	---
40	**2**	**0.85286**	**2**	**0.90024**
40	3	0.84538	3	---
40	**4**	**0.90274**	4	---

6. Usando Redes Pré-Treinadas da ImageNet e SVM

A etapa seguinte do experimento realizou extração de características para cada conjunto de treinamento (sem *data augmentation* e com *data augmentation*, seguindo as mesmas estratégias anteriores). Para isto, foi usada uma rede pré-treinada da *ImageNet* [Stanford Vision Lab 2016], a Xception [Keras SIG 2020b].

Depois, foi usado o classificador SVM da biblioteca SciKit Learn para testar. Apesar do tamanho padrão de entrada do Xception ser de 299x299, foi usado 32x32 para reduzir o tamanho do arquivo de característica e tempo de extração. Além disso, o tamanho 299x299 seria maior que a maioria dos exemplos. Mesmo com essa redução, o processo de extração de características para cada conjunto (1, 2 e 3) demorou aproximadamente 70 minutos e 130 minutos para o conjunto 4. O conjunto 0 (sem *data augmentation*) ficou com 26MB de tamanho, os conjuntos 1 2 e 3 com aproximadamente 320MB e o conjunto 4, com todos os exemplos gerados, 935GB. Devido ao tamanho, estes não estão juntados no projeto, mas podem ser gerados.

Os resultados são apresentados na Tabela 8. Percebe-se que a acurácia em qualquer uma das configurações ficou bem abaixo da alcançada pela LeNet5 e pela Outra CNN anteriormente demonstrada.

Tabela 8 - Resultados com Classificador SVM baseado em *Xception*

Conjunto	Acurácia
C0, sem Data Augmentation	0.22444
C1, com DA baseado em fontes	0.21945
C2, com DA por *Text-Image-Augmentation*	**0.30673**
C3, com DA de manipulação direta	0.28429

6.1. Comparando Matrizes de Confusão

Para finalizar, comparamos as matrizes de confusão para os melhores resultados da LeNet5, Outra CNN e para SVM.

Sem DA (Quadro 15), o melhor resultado foi da Outra CNN quando parametrizada com tamanho de 64x64 e 40 épocas. Percebe-se uma boa distribuição de poucos erros e um excelente nível de acerto, principalmente para os meses de março, abril e maio. Os principais erros foram a troca de janeiro por fevereiro e julho por junho. Na LeNet5, a principal confusão foi entre janeiro e fevereiro, julho por janeiro e dezembro por agosto. Já a matriz de confusão da SVM com extração de características feitas pela *Xception* revelou-se desastrosa. Acertando praticamente apenas os meses de março e abril. Março também foi o mês de maior acerto para a LeNet5.

Com DA (Quadro 16), considerando apenas os experimentos que usaram a base original acrescida a uma estratégia de DA, o melhor resultado foi também da Outra CNN, seguida por LeNet5. Nestas duas, a diagonal principal está bem destacada. Seus principais erros foram: para a LeNet5, a troca de abril por maio e dezembro por setembro ou agosto; para a Outra CNN, fevereiro por janeiro. A matriz de confusão da SVM mostrou-se mais uma vez desastrosa com bom acerto praticamente apenas março e abril.

Quadro 15 - Matrizes de Confusão para Execuções de Melhor Acurácia, sem DA

LeNet5 64x64 40 épocas Acc: 0.69326	Confusion Matrix: [[25 5 0 0 0 4 4 1 0 0 0 0] [7 20 0 2 1 0 0 1 1 0 0 0] [1 0 32 0 0 0 2 0 1 0 0 0] [3 1 1 25 4 0 0 2 1 1 1 0] [1 1 1 6 28 0 0 0 0 1 0 0] [4 0 0 0 0 17 4 3 0 1 0 0] [5 1 1 0 0 3 21 0 0 0 0 1] [0 0 0 0 0 0 0 21 0 0 1 6] [1 0 0 1 0 0 0 0 22 3 3 1] [0 0 2 1 0 0 2 0 1 24 0 0] [0 2 0 0 2 0 0 0 2 1 26 1] [0 1 0 0 1 1 0 2 5 1 5 17]]
Outra CNN 64x64 40 épocas Acc: 0.81795	Confusion Matrix: [[28 2 0 0 0 3 5 1 0 0 0 0] [6 20 0 1 2 0 1 1 0 0 1 0] [0 0 34 0 2 0 0 0 0 0 0 0] [0 0 0 36 2 0 0 0 0 1 0 0] [0 0 0 0 37 0 0 0 0 0 1 0] [4 0 0 0 0 17 7 0 0 0 1 0] [1 1 1 0 0 0 29 0 0 0 0 0] [0 0 1 0 0 0 0 23 0 0 1 3] [0 0 0 3 1 0 1 0 21 1 2 2] [0 0 1 0 0 0 1 0 0 28 0 0] [0 0 0 2 2 0 0 0 2 0 28 0] [0 0 0 0 0 2 0 1 1 1 1 27]]
SVM 32x32 Xception Acc: 0.22444	Confusion Matrix: [[8 3 21 3 1 1 1 1 0 0 0 0] [8 2 21 1 0 0 0 0 0 0 0 0] [3 0 29 2 0 0 1 0 0 1 0 0] [4 0 7 27 0 0 0 0 0 0 1 0] [2 0 16 7 10 0 0 0 1 0 2 0] [5 0 15 5 0 1 1 0 0 0 1 1] [9 0 10 8 2 0 2 1 0 0 0 0] [7 0 17 2 1 0 0 0 0 1 0 0] [5 1 14 8 0 0 0 0 1 0 2 0] [3 0 10 8 2 0 0 0 1 2 4 0] [2 1 15 3 4 0 0 0 1 0 8 0] [6 0 20 1 1 1 1 0 0 1 2 0]]

Quadro 16 - Matrizes de Confusão para Execuções de Melhor Acurácia, com DA

LeNet5 64x64 40 épocas C2 Acc: 0.86034	Confusion Matrix: [[34 3 0 0 0 1 1 0 0 0 0 0] [1 26 0 0 0 0 0 2 1 0 1 1] [0 0 35 1 0 0 0 0 0 0 0 0] [0 0 0 36 0 0 1 2 0 0 0 0] [0 0 0 5 32 1 0 0 0 0 0 0] [3 0 0 0 0 24 2 0 0 0 0 0] [0 0 0 0 1 4 27 0 0 0 0 0] [0 0 0 0 0 0 0 24 0 0 1 3] [0 0 0 0 0 0 0 0 24 1 2 4] [0 0 0 0 0 0 0 0 0 30 0 0] [0 1 0 0 0 0 0 0 2 0 31 0] [0 0 0 0 0 2 0 1 5 1 2 22]]
Outra CNN 64x64 40 épocas C2 Acc: 0.90024	Confusion Matrix: [[33 4 0 0 0 1 1 0 0 0 0 0] [2 28 0 0 0 0 1 0 0 1 0 0] [0 0 35 0 1 0 0 0 0 0 0 0] [0 0 0 37 1 0 0 1 0 0 0 0] [0 0 0 3 35 0 0 0 0 0 0 0] [3 0 0 0 0 24 2 0 0 0 0 0] [0 0 0 0 0 3 29 0 0 0 0 0] [0 0 1 0 0 0 0 25 0 0 0 2] [0 0 0 0 0 0 0 0 27 1 0 3] [0 0 1 0 0 0 0 0 0 29 0 0] [0 0 0 0 0 0 0 0 0 1 32 1] [0 0 0 0 0 1 0 1 1 0 3 27]]
SVM 32x32 C2 Xception Acc: 0.30673	Confusion Matrix: [[10 5 14 4 1 0 2 2 0 1 0 0] [4 6 15 1 1 0 0 2 1 0 0 2] [0 2 30 0 1 0 1 0 0 1 0 1] [1 1 7 25 0 0 0 0 2 1 1 1] [1 2 9 4 14 0 0 0 2 3 3 0] [4 0 15 2 0 3 3 1 0 0 0 1] [3 1 8 6 1 2 9 2 0 0 0 0] [3 1 16 1 1 0 0 3 0 1 0 2] [2 2 11 3 0 0 1 1 6 1 1 3] [2 1 8 5 3 0 0 0 4 3 2 2] [2 1 11 4 2 0 0 0 1 1 11 1] [4 0 15 0 1 2 2 0 3 1 2 3]]

7. Considerações Finais

Observa-se, ao final, que para o caso aqui tratado, o processo de *data augmentation* melhorou a acurácia das redes, principalmente quando a estratégia baseava-se em modificações sutis nas imagens da base original, aqui demonstradas pela biblioteca *Text-Image-Aumentation*, introduzida por Luo et a. [2020].

Usar uma rede pré-treinada para extrair características não trouxe bons resultados. Ao menos na escolha pela *Xception* com os parâmetros aqui apresentados. Devido ao demorado tempo de processamento para extração não conseguimos rodar outros cenários a fim de procurar melhor resultado.

Toda a implementação está disponível em repositório do GitHub: https://github.com/thiagotpc/ml-laboratorio-03, incluindo as saídas geradas pela execução.

Referências

Google LLC (2020). Google Fonts. Disponível em: https://fonts.google.com/. Acesso em 7set 2020.

Keras SIG (2020a). Keras: the Python deep learning API. Disponível em: https://keras.io/. Acesso em 8set 2020.

Keras SIG (2020b). Keras documentation: Xception. Disponível em: https://keras.io/api/applications/xception/. Acesso em 8set 2020.

LeCun, Y. (2020). MNIST Demos on Yann LeCun's website. Disponível em: http://yann.lecun.com/exdb/lenet/. Acesso em 8set 2020.

Luo, C. (2020). *Canjie-Luo/Text-Image-Augmentation*.

Luo, C., Zhu, Y., Jin, L. e Wang, Y. (2020). Learn to Augment: Joint Data Augmentation and Network Optimization for Text Recognition.

Python Software Foundation (2020). Pillow · PyPI. Disponível em: https://pypi.org/project/Pillow/. Acesso em 7set 2020.

RubanSeven (2020). *RubanSeven/Text-Image-Augmentation-python*.

Stanford Vision Lab (2016). ImageNet. Disponível em: http://www.image-net.org/. Acesso em 8set 2020.