UNIVERSIDADE CATÓLICA DE PELOTAS
Mestrado em Engenharia Eletrônica e Computação

Implementando uma Rede Neural para reconhecimento de dígitos manuscritos

Disciplina: Machine Learning

Pedro Tavares
Tarso Avila

Prof. Mateus Grellert

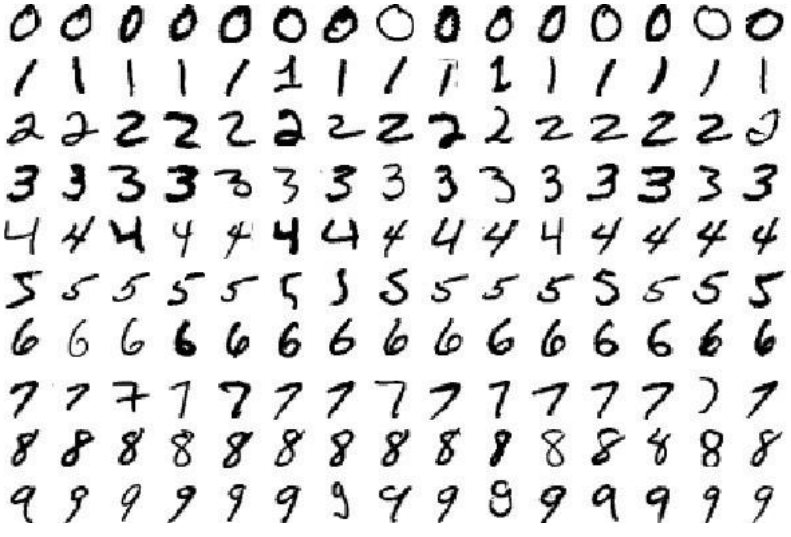
Pelotas, dezembro de 2019

Introdução

Classificar dígitos manuscritos a partir de imagens.

Utilizando:

- **Dataset: MNIST;**
- **Rede Neural Convolucional (CNN).**

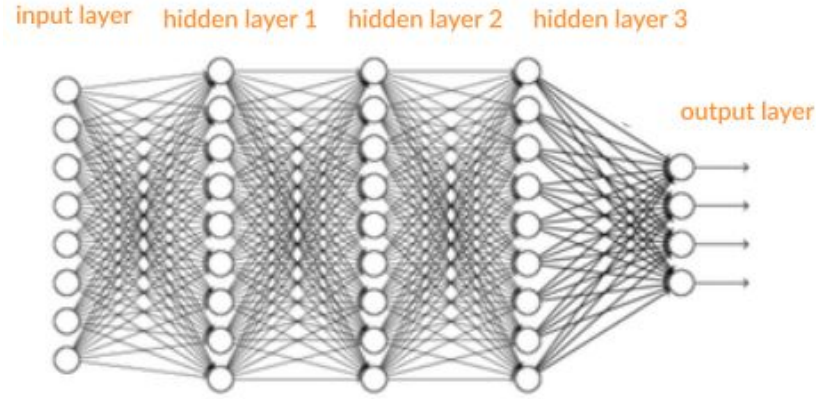


Introdução

Rede Neurais para classificação de imagens

As redes neurais clássicas são consideradas ineficientes para as tarefas de visão computacional, pois as imagens possuem um grande número de informações.

Para uma rede neural processar milhares de pixels e até 3 canais de cores precisa de um grande número de conexões e parâmetros para a rede.



Introdução

Rede Neural Convolucional (CNN)

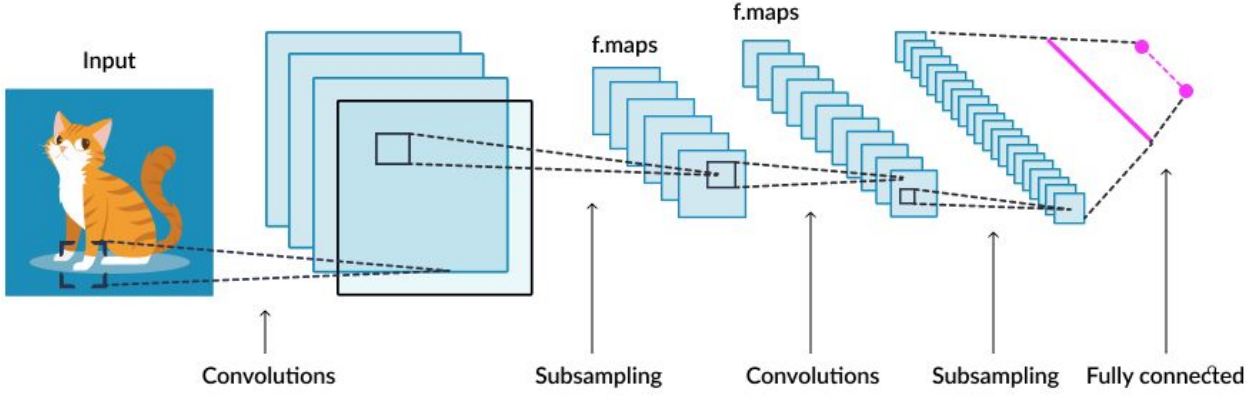
- Aproveita o fato que imagens são composta de detalhes (recursos);
- Recursos são extraídos por mecanismos criados pela rede;
- Recursos são analisados isoladamente;
- A partir dos recursos analisados isoladamente é possível ter uma decisão da imagem como um todo.

Obs.: Há uma camada deste tipo de rede que é totalmente conectada e por ela se obtém o resultado final de uma decisão/classificação.

Metodologia

Camadas fundamentais de uma Rede Neural Convolucional (CNN)

- Camada Convolucional;
- Camada de subamostragem;
- Camada totalmente conectada.



Metodologia

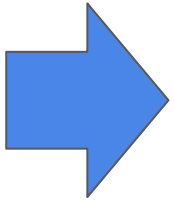
Topologia e parâmetros de uma CNN

Construção de um modelo CNN através API Keras (TensorFlow) usando Python

Passos:

- Remodelar a entrada de dados de acordo com a camada convolucional que se deseja usar;

- Criar camada convolucional (1d, 2d, 3d);
- Criar camada de Pooling;



Repetir passos para camadas adicionais

- Remodelar as saídas das camadas de convolução e pooling;
- Criar camada totalmente conectada;
- Criar camada final para previsão de classe.

Metodologia

Topologia e parâmetros de uma CNN

Parâmetros de uma camada de convolução (Conv2D), através API Keras (TensorFlow) usando Python:

- Filters;
- Kernel_size;
- Strides;
- Padding;
- Activation.

Em Python:

```
model.add(Conv2D(filters=48,kernel_size=5,strides=
1,padding='same',activation='relu'))
```

Metodologia

Encontrar a melhor modelo de CNN para classificar dados do dataset MNIST

Melhor arquitetura?

- No nosso caso a melhor arquitetura seria um compromisso entre simples, eficiente, precisa e com menor esforço computacional.

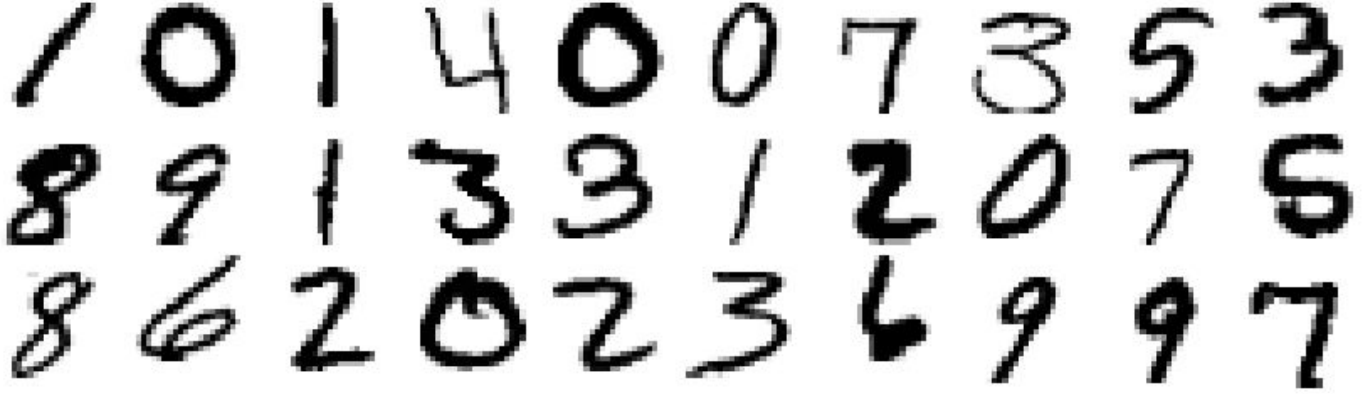
Como encontrar a melhor arquitetura?

- Experimentos;
- Métricas para avaliar treino e teste;
- Avaliações de desempenho.

Metodologia

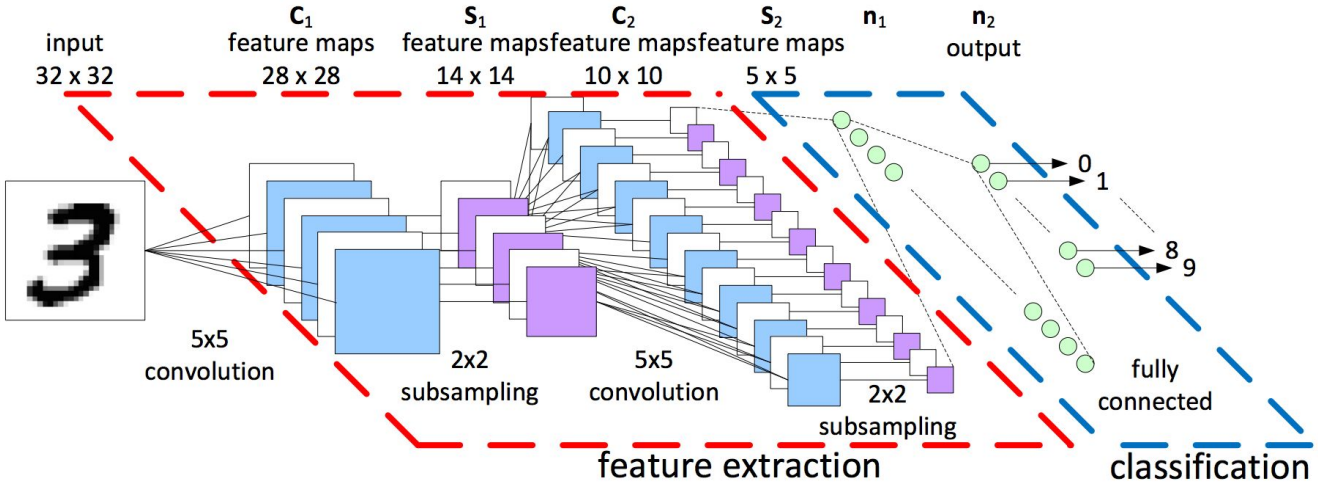
- O dataset é formado por imagens 28 x 28 pixels em tons de cinza (cada pixel possui um valor entre 0 e 255)

- O arquivo .csv original continha 42.000 amostras que foram divididas em:
  - 29.400 amostras para treino;
  - 12.600 amostras para teste;



Metodologia

A rede neural utilizada é baseada na LeNet-1, uma rede neural clássica para classificação profunda de imagens:



Metodologia

Topologia e parâmetros de uma CNN para reconhecimento de manuscritos

- Quantas camadas de convolução/subamostragem?
- Quantos mapas de recursos em cada camada de convolução?
- Quantos nós na camada totalmente conectada?
- Qual nível de *dropout* devemos utilizar?
- Qual o tamanho das camadas de convolução?
- Usar normalização em lote?
- Utilizar mecanismos para aumentar os dados?

Experimento 1

Camadas de convolução/subamostragem

Experimentar rede com 1, 2 e 3 camadas de convolução/subamostragem

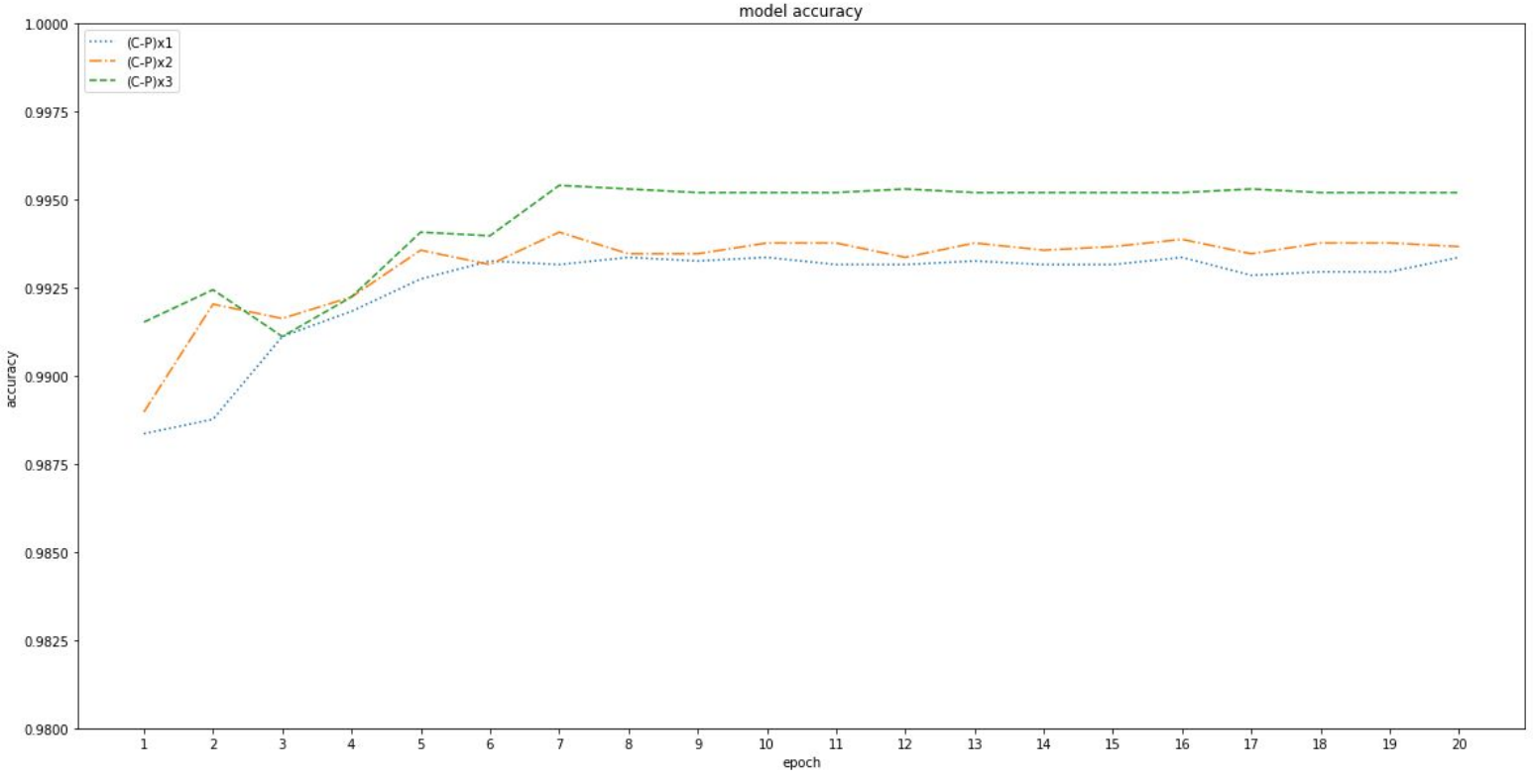
- 784 - **[24C5-P2]** - 256 - 10
- 784 - **[24C5-P2] - [48C5-P2]** - 256 - 10
- 784 - **[24C5-P2] - [48C5-P2] - [64C5-P2]** - 256 - 10

Experimento 1

```
# BUILD CONVOLUTIONAL NEURAL NETWORKS
nets = 3
model = [0] *nets

for j in range(3):
    model[j] = Sequential()
    model[j].add(Conv2D(24,kernel_size=5,padding='same',activation='relu',
            input_shape=(28,28,1)))
    model[j].add(MaxPool2D())
    if j>0:
        model[j].add(Conv2D(48,kernel_size=5,padding='same',activation='relu'))
        model[j].add(MaxPool2D())
    if j>1:
        model[j].add(Conv2D(64,kernel_size=5,padding='same',activation='relu'))
        model[j].add(MaxPool2D(padding='same'))
    model[j].add(Flatten())
    model[j].add(Dense(256, activation='relu'))
    model[j].add(Dense(10, activation='softmax'))
    model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 1 - Resultado



Experimento 1 Resultado

Results:

CNN (C-P)x1: Epochs=20, Train accuracy=1.00000, Validation accuracy=0.99336

CNN (C-P)x2: Epochs=20, Train accuracy=1.00000, Validation accuracy=0.99408

CNN (C-P)x3: Epochs=20, Train accuracy=1.00000, Validation accuracy=0.99540

Experimento 2

O resultado do experimento anterior mostrou que com duas camadas de convolução é o suficiente. Experimentar quantidade de mapas de recurso para cada camada.

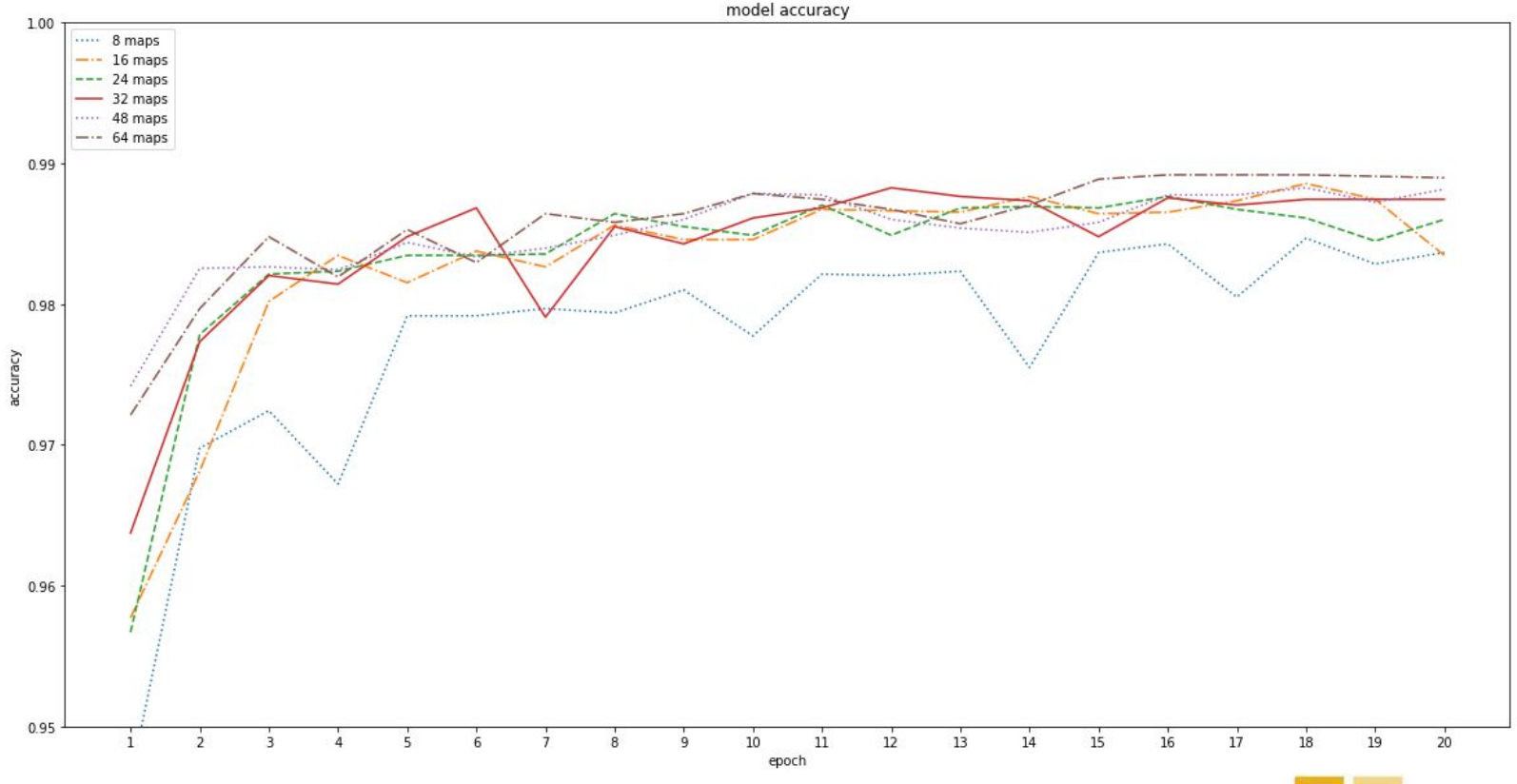
- 784 - [**8**C5-P2] - [**16**C5-P2] - 256 - 10
- 784 - [**16**C5-P2] - [**32**C5-P2] - 256 - 10
- 784 - [**24**C5-P2] - [**48**C5-P2] - 256 - 10
- 784 - [**32**C5-P2] - [**64**C5-P2] - 256 - 10
- 784 - [**48**C5-P2] - [**96**C5-P2] - 256 - 10
- 784 - [**64**C5-P2] - [**128**C5-P2] - 256 - 10

Experimento 2

```
# BUILD CONVOLUTIONAL NEURAL NETWORKS
nets = 6
model = [0] *nets

for j in range(6):
    model[j] = Sequential()
    model[j].add(Conv2D(j*8+8,kernel_size=5,activation='relu',input_shape=(28,28,1)))
    model[j].add(MaxPool2D())
    model[j].add(Conv2D(j*16+16,kernel_size=5,activation='relu'))
    model[j].add(MaxPool2D())
    model[j].add(Flatten())
    model[j].add(Dense(256, activation='relu'))
    model[j].add(Dense(10, activation='softmax'))
    model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 2 - Resultado



Experimento 2 - Resultado

Results:

CNN 8 maps: Epochs=20, Train accuracy=0.99918, Validation accuracy=0.98468

CNN 16 maps: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98856

CNN 24 maps: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98764

CNN 32 maps: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98825

CNN 48 maps: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98825

CNN 64 maps: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98917

Experimento 3

O resultado do experimento anterior mostrou que utilizando 32 e 64 mapas de recursos na primeira e segunda camada de convolução, respectivamente, é o ideal. Experimentar o número de nós na camada totalmente conectada.

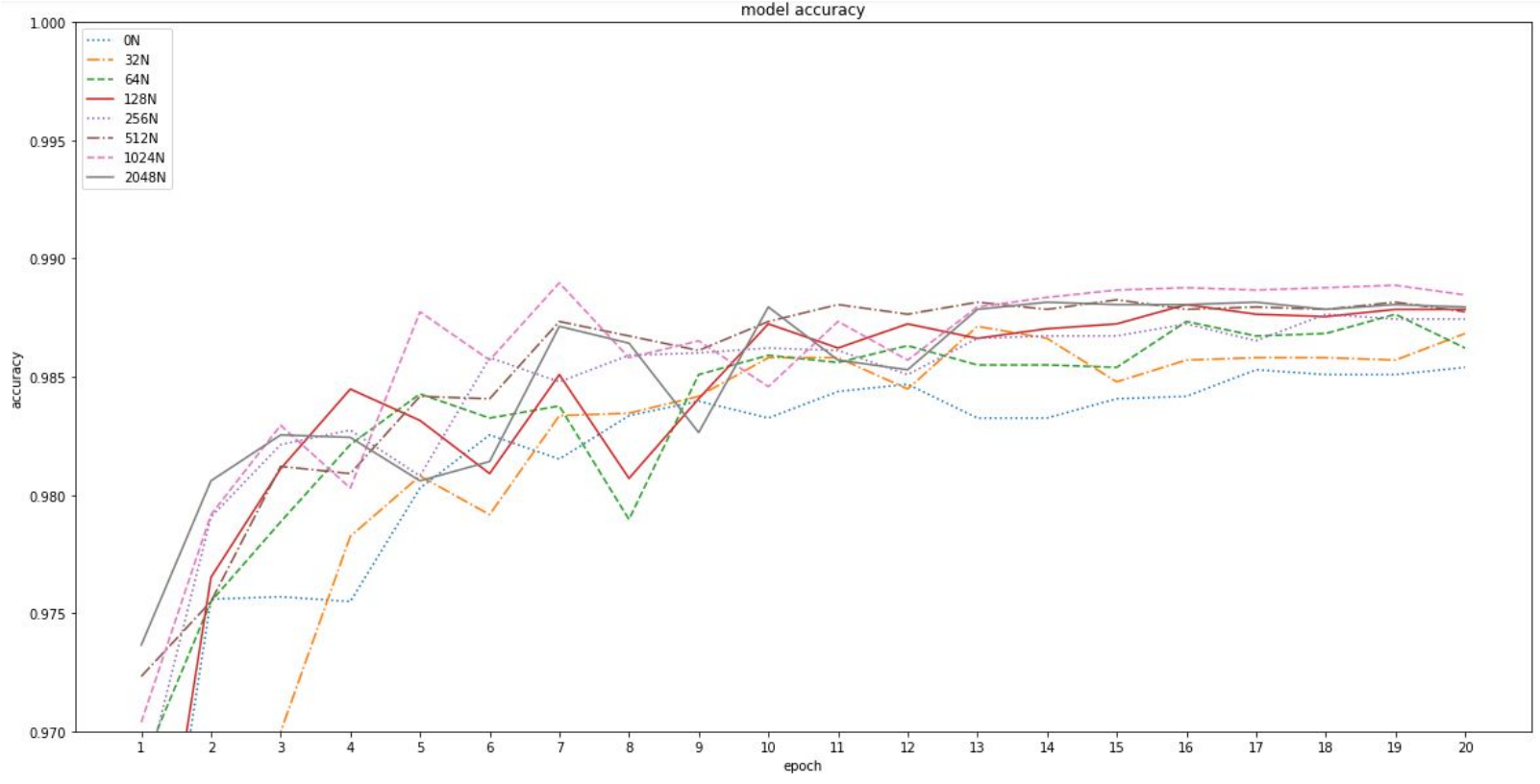
- 784 - [32C5-P2] - [64C5-P2] - **0** - 10
- 784 - [32C5-P2] - [64C5-P2] - **32** - 10
- 784 - [32C5-P2] - [64C5-P2] - **64** - 10
- 784 - [32C5-P2] - [64C5-P2] - **128** -10
- 784 - [32C5-P2] - [64C5-P2] - **256** - 10
- 784 - [32C5-P2] - [64C5-P2] - **512** -10
- 784 - [32C5-P2] - [64C5-P2] - **1024** - 10
- 784 - [32C5-P2] - [64C5-P2] - **2048** - 10

Experimento 3

```
# BUILD CONVOLUTIONAL NEURAL NETWORKS
nets = 8
model = [0] *nets

for j in range(8):
    model[j] = Sequential()
    model[j].add(Conv2D(32,kernel_size=5,activation='relu',input_shape=(28,28,1)))
    model[j].add(MaxPool2D())
    model[j].add(Conv2D(64,kernel_size=5,activation='relu'))
    model[j].add(MaxPool2D())
    model[j].add(Flatten())
    if j>0:
        model[j].add(Dense(2**(j+4), activation='relu'))
    model[j].add(Dense(10, activation='softmax'))
    model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 3 - Resultado



Experimento 3 - Resultado

Results:

CNN 0N: Epochs=20, Train accuracy=0.99985, Validation accuracy=0.98539

CNN 32N: Epochs=20, Train accuracy=0.99985, Validation accuracy=0.98713

CNN 64N: Epochs=20, Train accuracy=0.99990, Validation accuracy=0.98764

CNN 128N: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98805

CNN 256N: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98764

CNN 512N: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98825

CNN 1024N: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98897

CNN 2048N: Epochs=20, Train accuracy=0.99995, Validation accuracy=0.98815

Experimento 4

O resultado do experimento anterior mostrou que com 128 nós o resultado é satisfatório. Experimentar o índice de dropout.

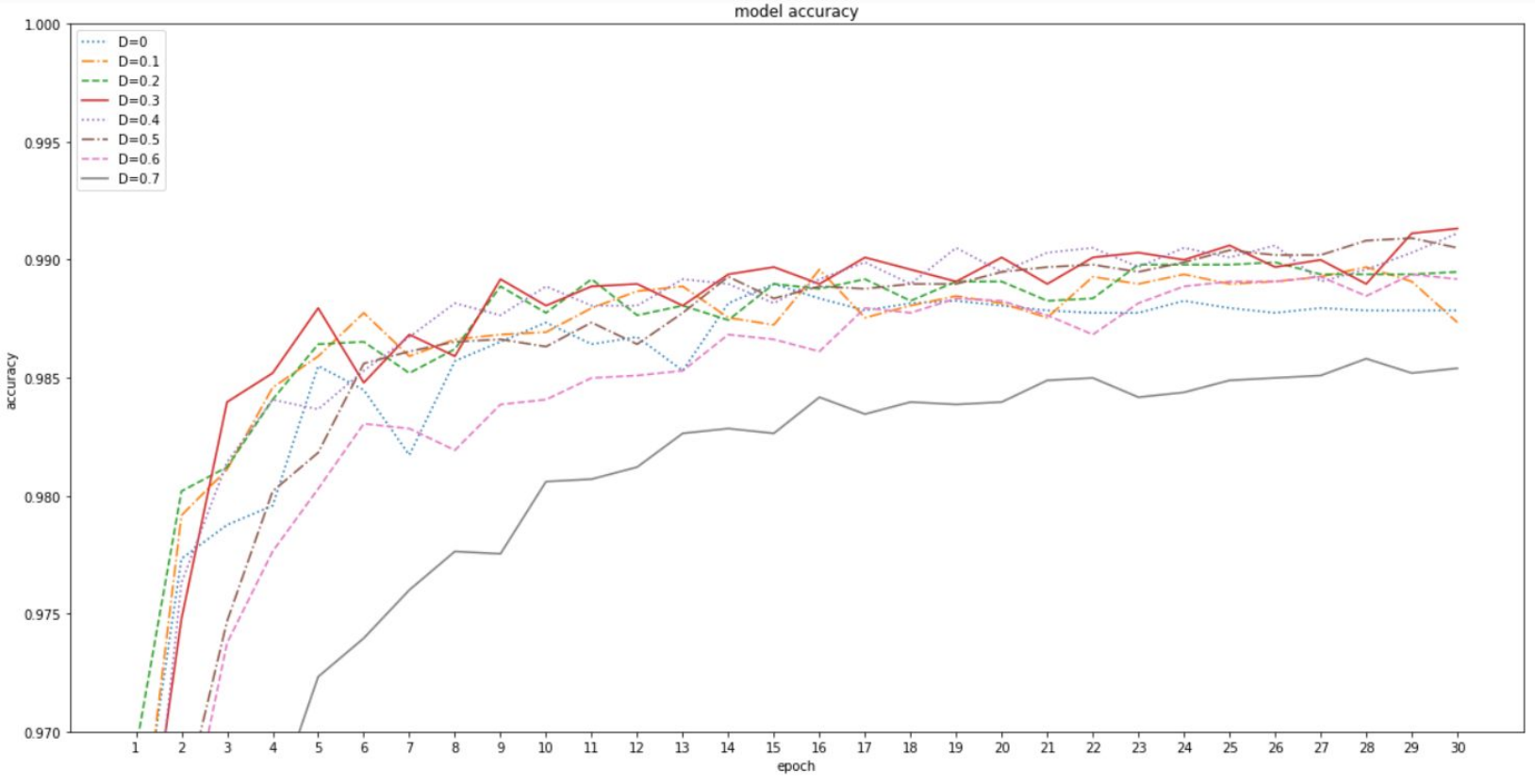
- **0%;**
- **10%;**
- **20%;**
- **30%;**
- **40%;**
- **50%;**
- **60%;**
- **70%.**

Experimento 4

```
# BUILD CONVOLUTIONAL NEURAL NETWORKS
nets = 8
model = [0] *nets

for j in range(8):
    model[j] = Sequential()
    model[j].add(Conv2D(32,kernel_size=5,activation='relu',input_shape=(28,28,1)))
    model[j].add(MaxPool2D())
    model[j].add(Dropout(j*0.1))
    model[j].add(Conv2D(64,kernel_size=5,activation='relu'))
    model[j].add(MaxPool2D())
    model[j].add(Dropout(j*0.1))
    model[j].add(Flatten())
    model[j].add(Dense(128, activation='relu'))
    model[j].add(Dropout(j*0.1))
    model[j].add(Dense(10, activation='softmax'))
    model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 4 - Resultado



Experimento 4 - Resultado

Results:

CNN D=0: Epochs=30, Train accuracy=0.99995, Validation accuracy=0.98897

CNN D=0.1: Epochs=30, Train accuracy=0.99959, Validation accuracy=0.98968

CNN D=0.2: Epochs=30, Train accuracy=0.99903, Validation accuracy=0.98989

CNN D=0.3: Epochs=30, Train accuracy=0.99618, Validation accuracy=0.99132

CNN D=0.4: Epochs=30, Train accuracy=0.99291, Validation accuracy=0.99111

CNN D=0.5: Epochs=30, Train accuracy=0.98812, Validation accuracy=0.99091

CNN D=0.6: Epochs=30, Train accuracy=0.97741, Validation accuracy=0.98938

CNN D=0.7: Epochs=30, Train accuracy=0.95946, Validation accuracy=0.98580

Experimento 5

Melhor rede até agora:

- 784 - [32C5-P2] - [64C5-P2] - 128 - 10 (com Dropout 40%)

Técnicas avançadas:

- 784 - [**32C3-32C3**-P2] - [**64C3-64C3**-P2] - 128 - 10
- 784 - [32C3-32C3-**32C5S2**] - [64C3-64C3-**64C5S2**] - 128 - 10
- + Batch normalization
- + Data augmentation

Experimento 5

```
# BUILD CONVOLUTIONAL NEURAL NETWORKS
nets = 5
model = [0] *nets

j=0
model[j] = Sequential()
model[j].add(Conv2D(32,kernel_size=5,activation='relu',input_shape=(28,28,1)))
model[j].add(MaxPool2D())
model[j].add(Dropout(0.4))
model[j].add(Conv2D(64,kernel_size=5,activation='relu'))
model[j].add(MaxPool2D())
model[j].add(Dropout(0.4))
model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))
model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 5

```
j=1
model[j] = Sequential()
model[j].add(Conv2D(32,kernel_size=3,activation='relu',input_shape=(28,28,1)))
model[j].add(Conv2D(32,kernel_size=3,activation='relu'))
model[j].add(MaxPool2D())
model[j].add(Dropout(0.4))
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(MaxPool2D())
model[j].add(Dropout(0.4))
model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))
model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 5

```
j=2
model[j] = Sequential()
model[j].add(Conv2D(32,kernel_size=5,activation='relu',input_shape=(28,28,1)))
model[j].add(Conv2D(32,kernel_size=3,activation='relu'))
model[j].add(Conv2D(32,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Conv2D(64,kernel_size=5,activation='relu'))
model[j].add(Conv2D(64,kernel_size=3,activation='relu'))
model[j].add(Conv2D(64,kernel_size=5,strides=2,padding='same',activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Flatten())
model[j].add(Dense(128, activation='relu'))
model[j].add(Dropout(0.4))
model[j].add(Dense(10, activation='softmax'))
model[j].compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

Experimento 5

```
j=3
model[j] = Sequential()

model[j].add(Conv2D( 32,kernel_size= 3,activation= 'relu',input_shape=( 28,28,1)))
model[j].add(BatchNormalization())
model[j].add(Conv2D( 32,kernel_size= 3,activation= 'relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D( 32,kernel_size= 5,strides= 2,padding= 'same',activation= 'relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout( 0.4))

model[j].add(Conv2D( 64,kernel_size= 3,activation= 'relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D( 64,kernel_size= 3,activation= 'relu'))
model[j].add(BatchNormalization())
model[j].add(Conv2D( 64,kernel_size= 5,strides= 2,padding= 'same',activation= 'relu'))
model[j].add(BatchNormalization())
model[j].add(Dropout( 0.4))

model[j].add(Flatten())
model[j].add(Dense( 128, activation= 'relu'))
model[j].add(Dropout( 0.4))
model[j].add(Dense( 10, activation= 'softmax'))

model[j].compile(optimizer= "adam", loss= "categorical_crossentropy" , metrics=[ "accuracy"])
```

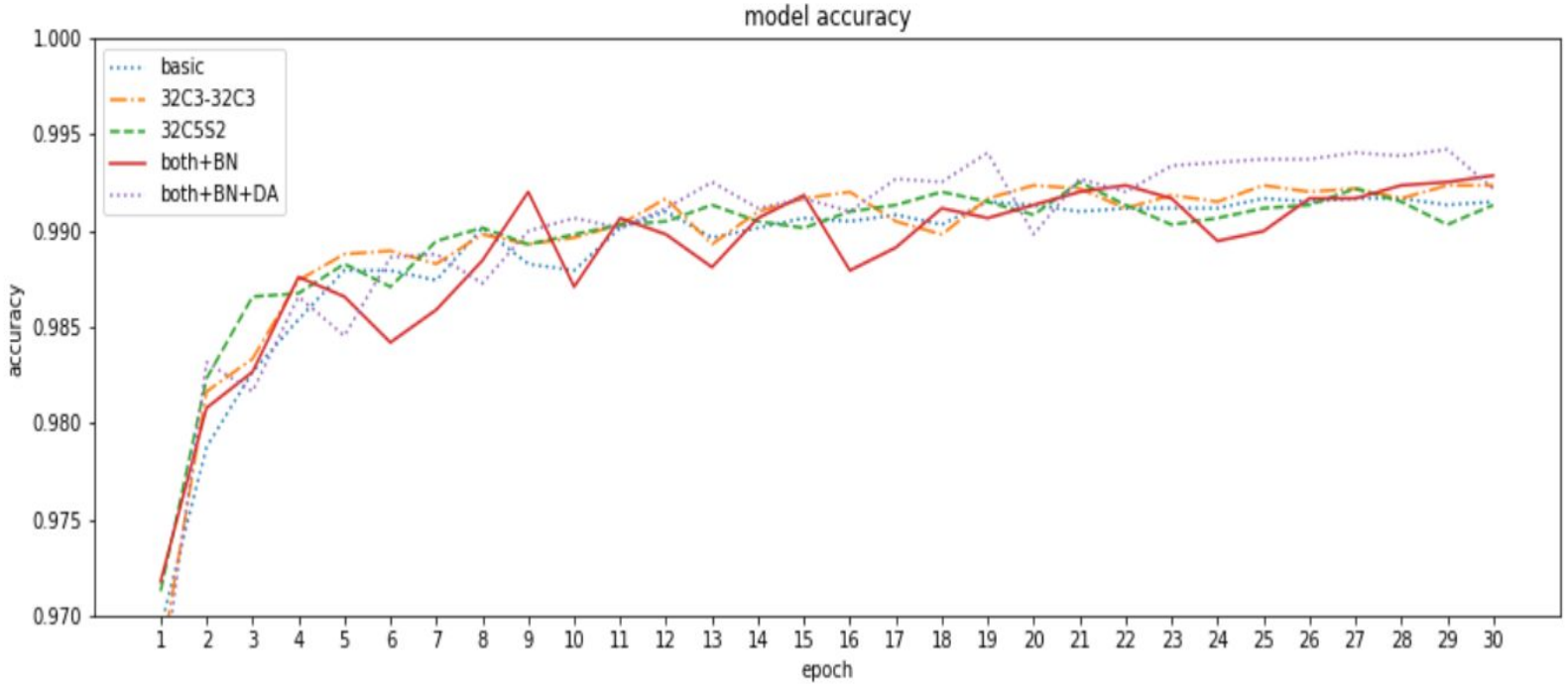
Experimento 5

```
j=4

#(...)
# SAME AS PREVIOUS MODEL

# CREATE MORE TRAINING IMAGES VIA DATA AUGMENTATION
datagen = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1)
```

Experimento 5 - Resultado



Experimento 5 - Resultado

Results:

CNN basic: Epochs=30, Train accuracy=0.99439, Validation accuracy=0.99167

CNN 32C3-32C3: Epochs=30, Train accuracy=0.99588, Validation accuracy=0.99235

CNN 32C5S2: Epochs=30, Train accuracy=0.99868, Validation accuracy=0.99252

CNN both+BN: Epochs=30, Train accuracy=0.99821, Validation accuracy=0.99286

CNN both+BN+DA: Epochs=30, Train accuracy=0.99297, Validation accuracy=0.99422

Experimento 5 - Resultado

EVALUATION WITH TEST.CSV:

CNN basic:
12600/12600 [==============================] - 5s 363us/step
Test accuracy: **0.9924603174603175** Test loss 0.02753697095508595

CNN 32C3-32C3:
12600/12600 [==============================] - 8s 620us/step
Test accuracy: **0.993015873015873** Test loss 0.02817932214556051

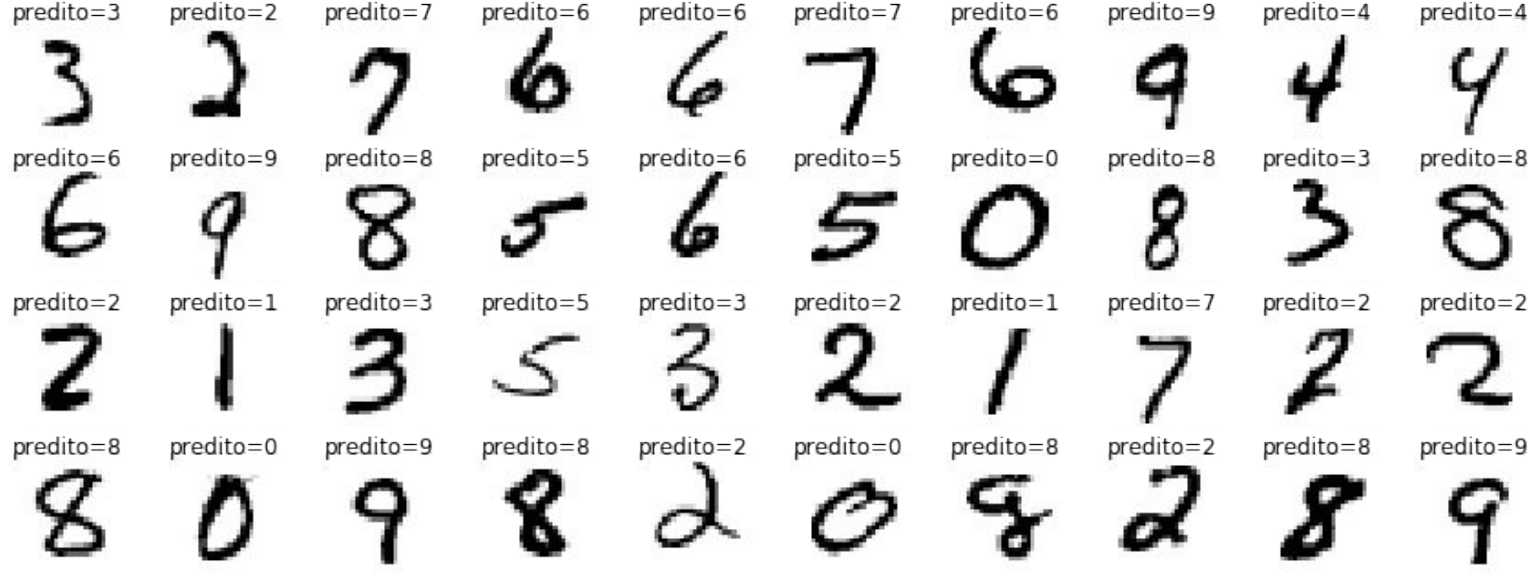
CNN 32C5S2:
12600/12600 [==============================] - 7s 562us/step
Test accuracy: **0.991984126984127** Test loss 0.03784768203479311

CNN both+BN:
12600/12600 [==============================] - 18s 1ms/step
Test accuracy: **0.9937301587301587** Test loss 0.03225840429869084

CNN both+BN+DA:
12600/12600 [==============================] - 18s 1ms/step
Test accuracy: **0.9942857142857143** Test loss 0.022522135234298368

Experimento 5 - Resultado

Classificação das 40 primeiras imagens do dataset de testes (para models j=0 e j=4):



1 erro (item[37]): acurácia 0,975

Conclusão

- Melhor rede considerando precisão e performance:

CNN basic:

784 - [32C5-P2] - [64C5-P2] - 128 - 10 (com Dropout 40%)

- Melhor rede considerando apenas precisão:

CNN both BN+DA:

784 - [32C3-32C3-32C5S2] - [64C3-64C3-64C5S2] - 128 - 10

Com Dropout 40% +
Batch Normalization +
Data Augmentation

Referências

- How to choose CNN Architecture MNIST, Kaggle. Disponível em: https://www.kaggle.com/kernels/scriptcontent/5965574/output, consulta em: 06/12/2019.
- Entendendo Redes Convolucionais (CNNs). Disponível em: https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184, consulta em: 05/12/2019.
- Review: LeNet-1, LeNet-4, LeNet-5, Boosted LeNet-4 (Image Classification), Disponível em: https://medium.com/@sh.tsang/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1f5f809dbf17, consulta em: 05/12/2019.
- LeNet – Convolutional Neural Network in Python, Disponível em: https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/, consulta em: 05/12/2019.