Relatório do Lab 9 de CT-213

Trabalho 9 – Redes Neurais Convolucionais

Reconhecendo escrita manual com a rede neural LeNET-5 e o *dataset* MNIST

Aluno:

Bruno Costa Alves Freire

Turma:

T 22.4

Professor:

Marcos Ricardo Omena de Albuquerque Máximo

Data:

24/05/2019

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Implementando a LeNET-5

A implementação da LeNET-5 segue a arquitetura descrita no roteiro do laboratório. Temos três camadas de convolução 2D, intercaladas com duas camadas de *average pooling* 2D, e duas camadas densas (*fully connected*) no final.

As camadas de convolução têm todas um *kernel* 5x5 com *stride* 1, enquanto as camadas de *pooling* usam um *pool* 2x2 com *stride* 2. A entrada da rede neural são imagens 32x32 (altura x largura) com 1 canal de cor. A quantidade de filtros nas camadas de convolução é de 6 na primeira, 16 na segunda e 120 na terceira. Devido aos tamanhos de *kernel*, *pool* e *strides* das 5 primeiras camadas, a saída para a primeira camada densa tem formato 1x1, razão pela qual colocamos uma camada de transição nesse ponto do tipo *flatten*.

As duas camadas densas contam respectivamente com 84 e 10 neurônios. Em todas as camadas a função de ativação é do tipo tangente hiperbólica, exceto pela última, a qual utiliza a função *softmax*. Ao todo, temos 61706 parâmetros a serem otimizados.

A implementação dessa arquitetura por meio do Keras requer apenas algumas linhas de código constituídas simplesmente por chamadas à função `model.add()`, passando os devidos parâmetros. O modelo da rede é construído na função `make_lenet5()` do *script* `lenet5.py`, e é retornado aos *scripts* que farão uso da rede neural.

2. Treinamento da LeNET-5

Para treinar a LeNET-5, fazemos o download do dataset MNIST, que conta com diversas imagens de caracteres de algarismos manuscritos. O *script* `train_lenet5.py` faz uma separação dos elementos desse dataset em um conjunto de treinamento e um de validação. Temos 48000 imagens no conjunto de treinamento e 12000 no de validação. A ideia na verdade é dividir o dataset em três conjuntos, o de treinamento propriamente dito, o de validação, e finalmente o de teste. O objetivo do conjunto de validação é tentar corrigir o problema do *overfitting*, atuando como uma espécie de conjunto de teste durante o próprio treinamento. Em outras palavras, a validação é um quiz, e o teste é a prova.

Após separar o dataset, inicia-se o treinamento da rede. O seu desempenho pode ser acompanhado através do TensorBoard, ativado por meio do *script* `run_tensorboard.py`. Da interface do TensorBoard foram obtidos os gráficos das figuras de 1 a 4.

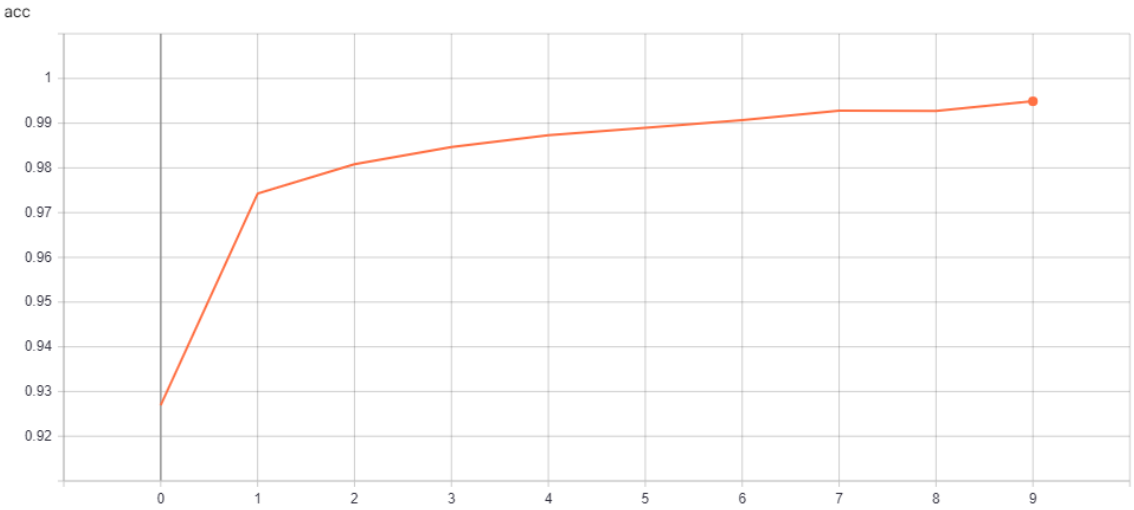

Figura 1: Evolução da acurácia da rede no conjunto de treinamento ao longo das épocas.

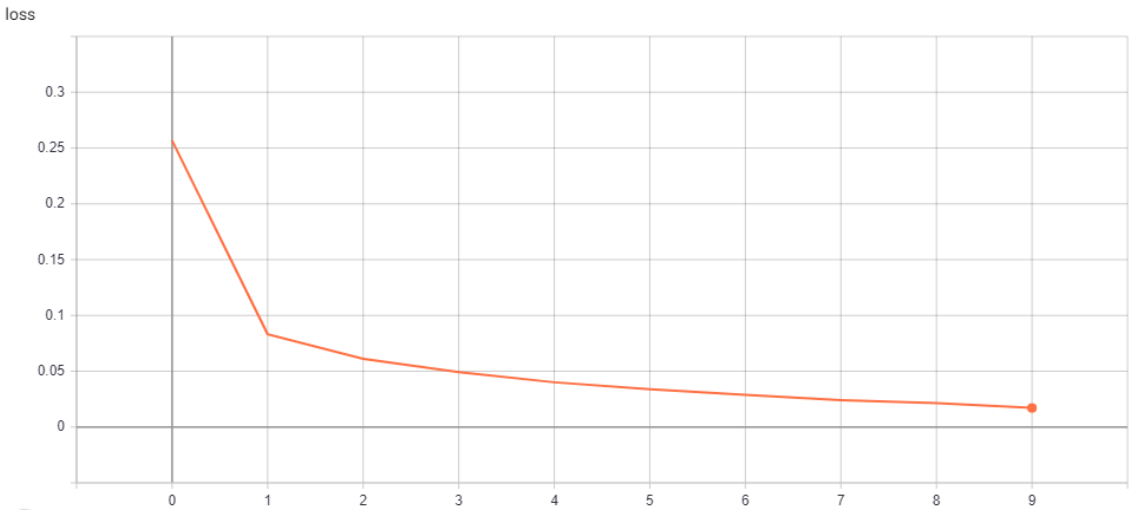

Figura 2: Evolução da função de custo aprendida pela rede aplicada ao conjunto de treinamento ao longo das épocas.

Das figuras 1 e 2 podemos observar o aprimoramento do aprendizado da rede, dado que a função de custo diminui, e a acurácia aumenta. No entanto, como não implementamos mecanismos de regularização, temos que nos preocupar com o problema do *overfitting*.

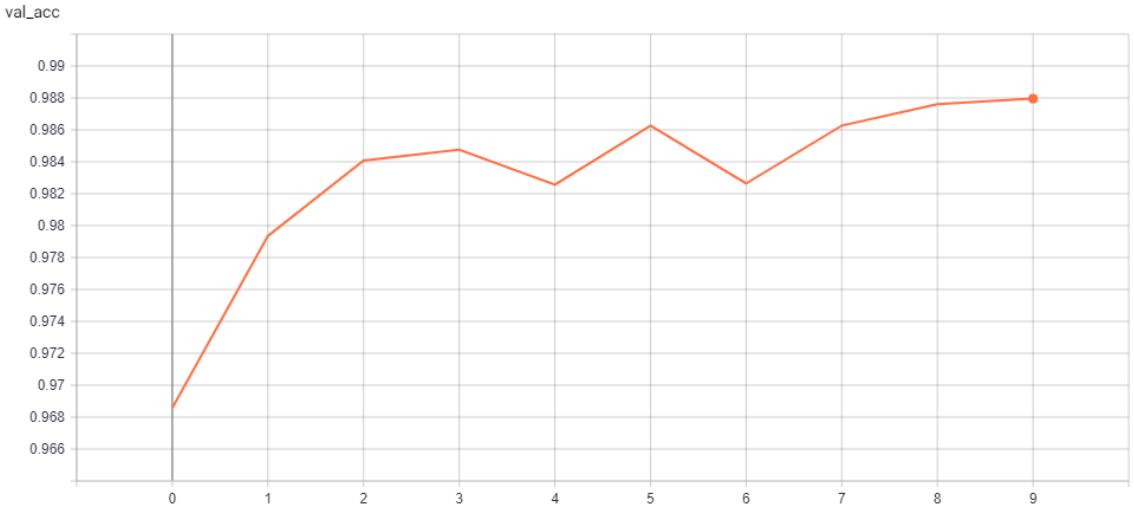

Figura 3: Evolução da acurácia da rede no conjunto de validação ao longo das épocas.

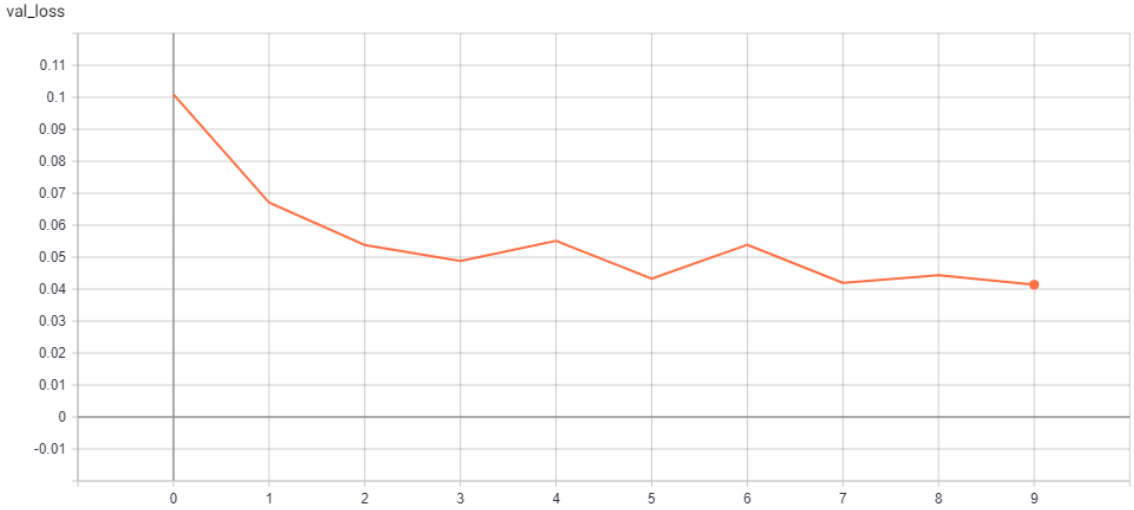

Figura 4: Evolução da função de custo aprendida pela rede aplicada ao conjunto de validação ao longo das épocas.

Olhando para as curvas das figuras 3 e 4, podemos ver que o aspecto não monotônico do gráfico indica que a rede errou bastante nas épocas 4 e 6, o que indica que provavelmente ela aprendeu algum aspecto incorreto da grafia dos dígitos durante o treinamento (“overfitou”), e recebeu um feedback negativo acerca desse aspecto nessas épocas.

3. Avaliação da LeNET-5

Após treinarmos a rede, vamos testar o seu desempenho com o restante do dataset, o conjunto de teste, que consiste de 10000 imagens de dígitos manuscritos.

Algumas métricas do desempenho da rede que pudemos extrair foram a função de custo final da rede, que foi de 0.03847468903241679, e a acurácia da rede no conjunto de teste, que foi de 98.82%, um desempenho bastante positivo, se levarmos em consideração a grafia esquisita de alguns dígitos do dataset.

Para verificar o desempenho da rede no conjunto, vamos tomar alguns exemplos de dígitos que a rede reconheceu corretamente, nas figuras de 5 a 9, e alguns onde a rede errou, nas figuras de 10 a 14.

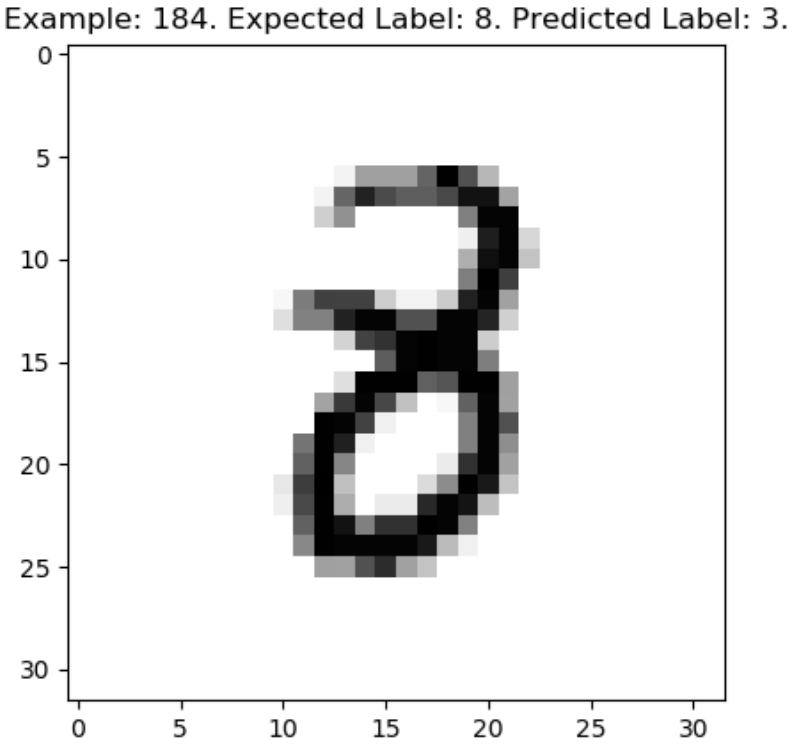

Figura 5: Exemplo de dígito que a rede identificou incorretamente.

Podemos notar que o fato da curva do 8 não estar fechada confundiu a rede, que o classificou como um 3, apesar de não ter a abertura na curva de baixo, como um bom 3 deveria ter.

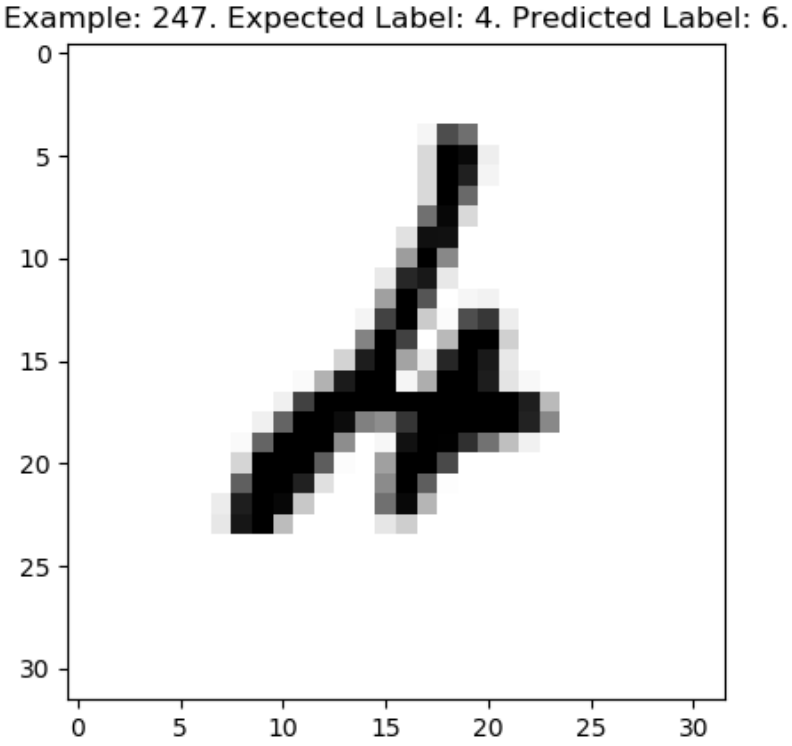

Figura 6: Exemplo de dígito que a rede identificou incorretamente.

Um 4 bastante esquisito, com traçados em ângulos típicos de um 6, mas sem fechar a curva de baixo, não parece nada com um 6. Apesar de que só de longe isso parece um 4.

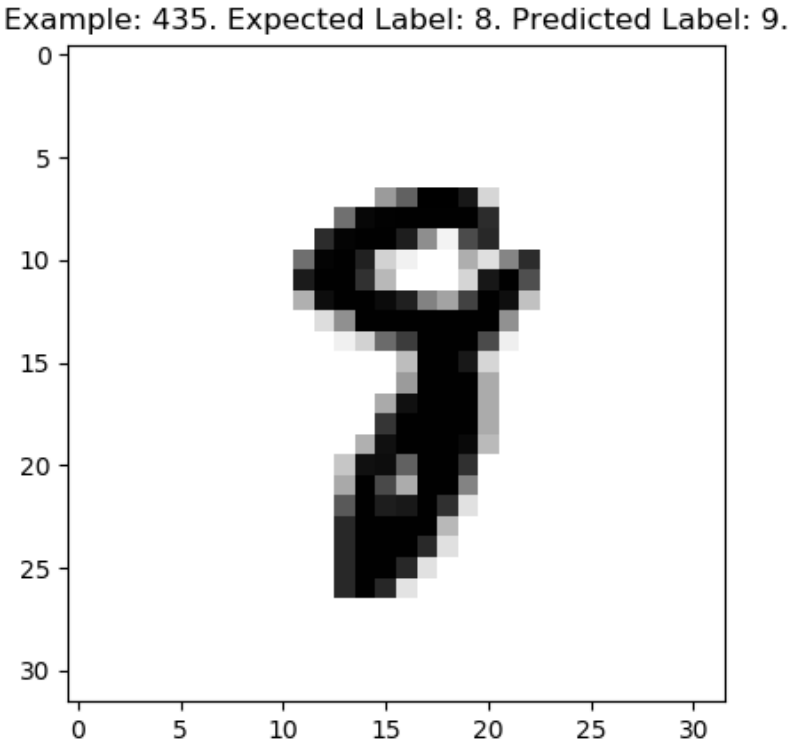

Figura 7: Exemplo de dígito que a rede identificou incorretamente.

Um 8 com a parte inferior estreita, quase colapsada para um traço único.

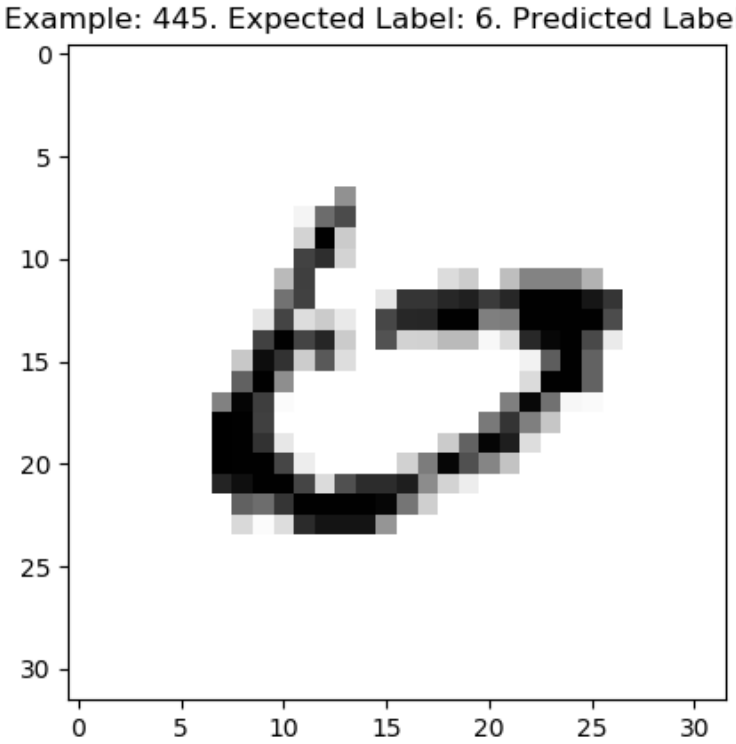

Figura 8: Exemplo de dígito que a rede identificou incorretamente.

Talvez esse seja o exemplo onde o erro da rede é mais compreensível. Trata-se de uma grafia bem preguiçosa de um 6, com uma desproporção na parte circular do dígito, que o levou a ser confundido com um 0.

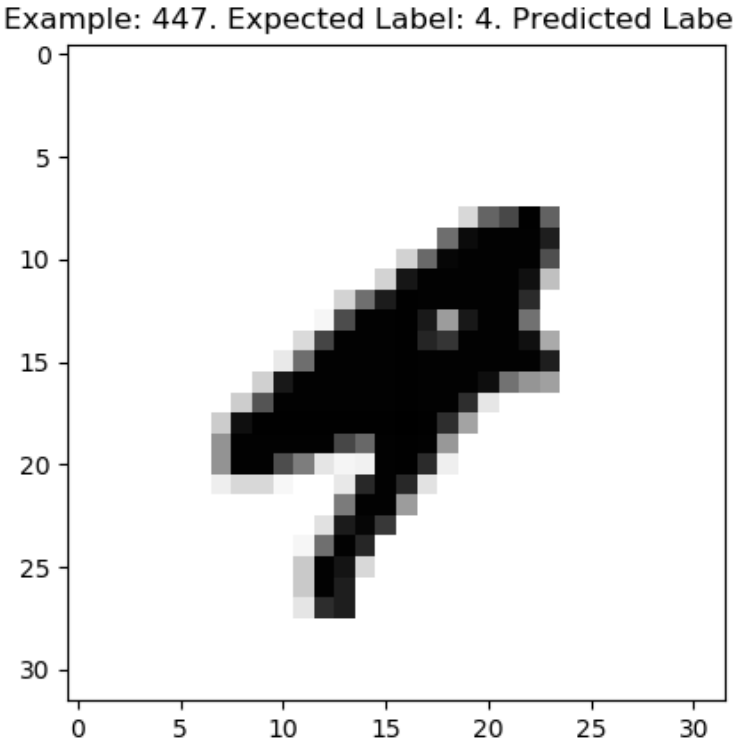

Figura 9: Exemplo de dígito que a rede identificou incorretamente.

Por fim, um 4 que poderia ser confundido com um 9 por qualquer um.

Em resumo, vimos que algumas grafias que não destacam bem alguns dos aspectos principais do formato dos algarismos são difíceis de reconhecer para a rede. Vejamos agora alguns dos exemplos em que a rede acertou.

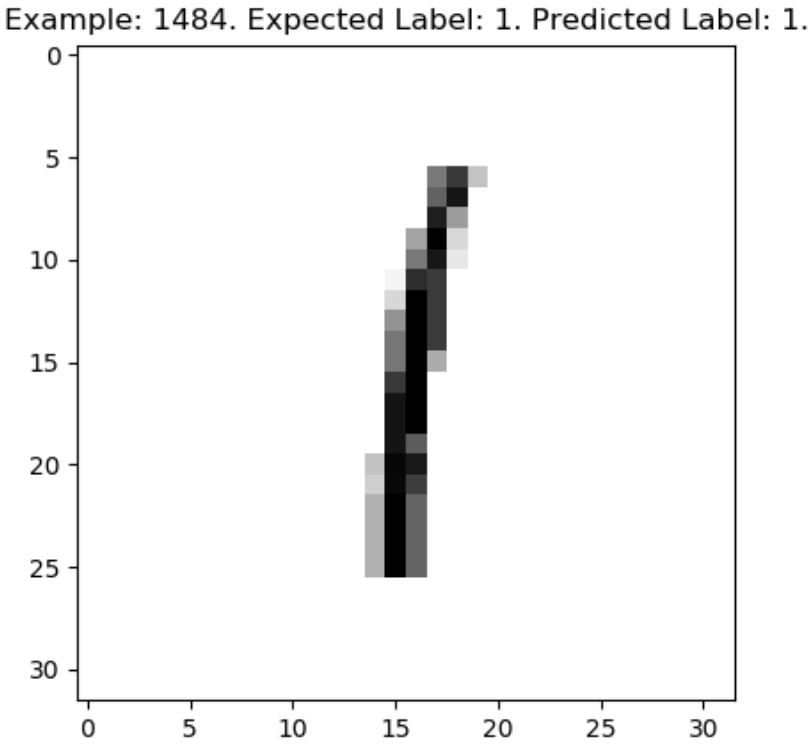

Figura 10: Exemplo de dígito que a rede identificou corretamente.

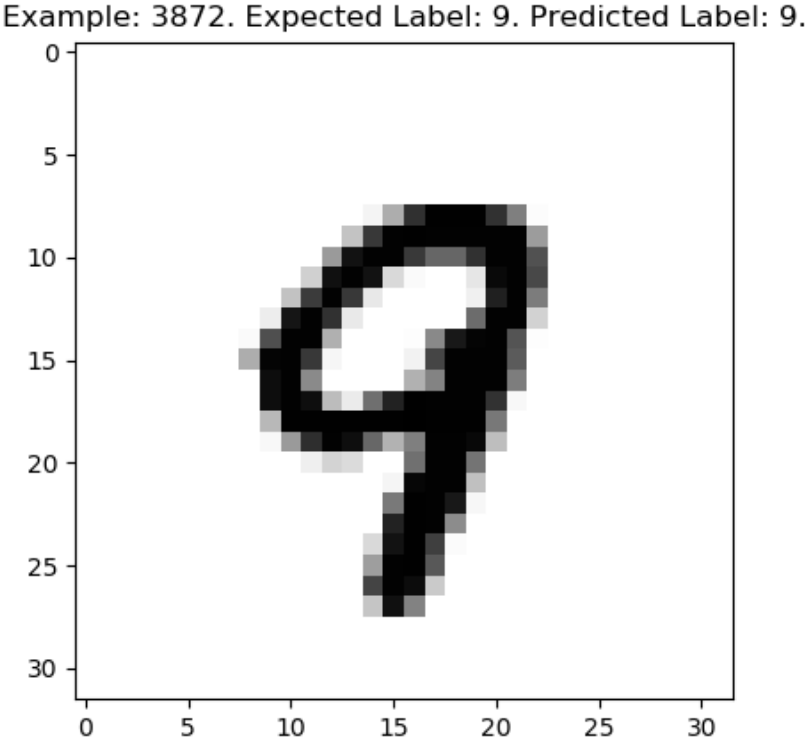

Figura 11: Exemplo de dígito que a rede identificou corretamente.

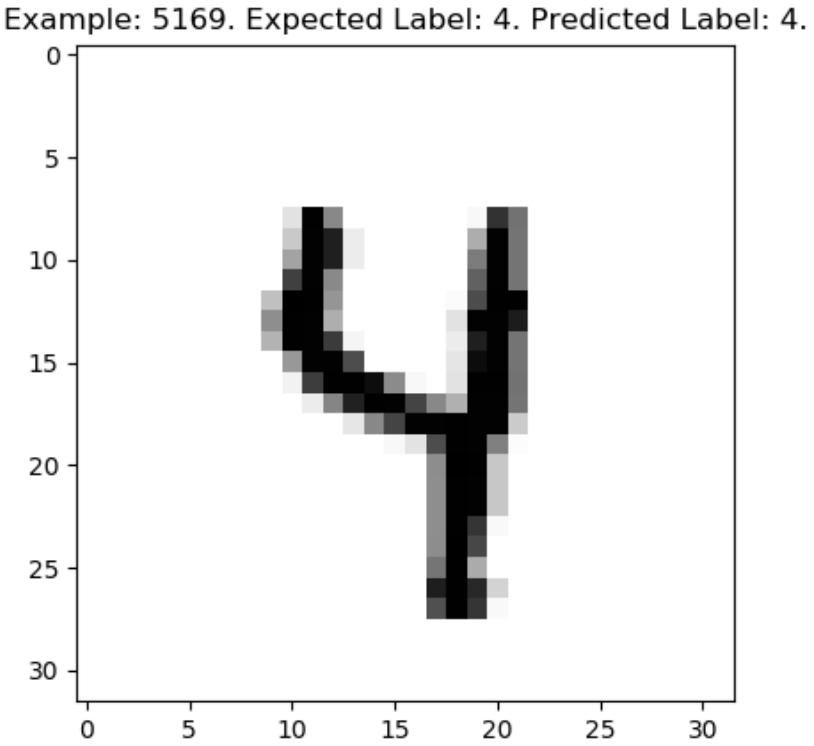

Figura 12: Exemplo de dígito que a rede identificou corretamente.

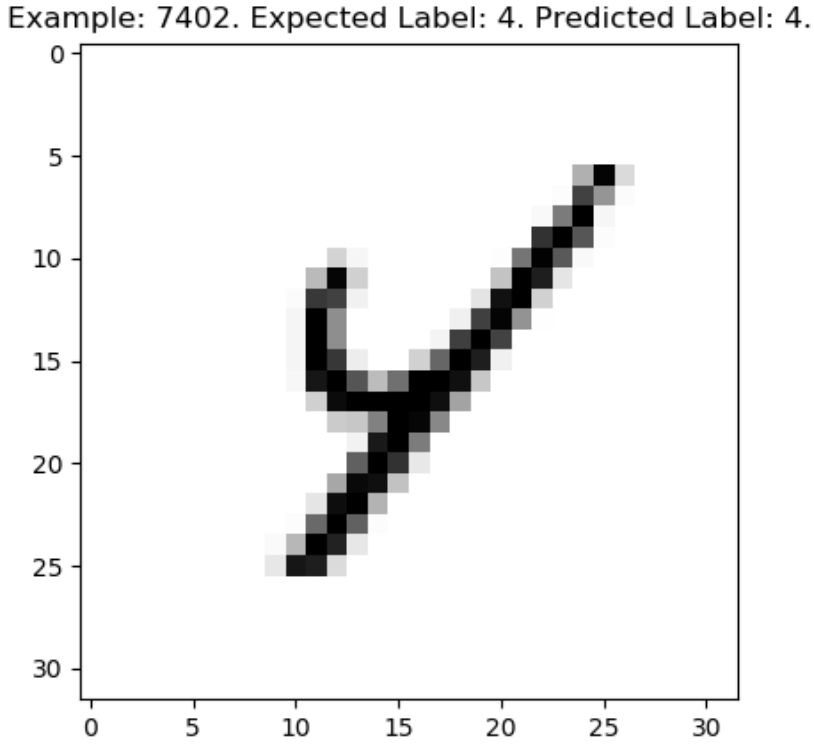

Figura 13: Exemplo de dígito que a rede identificou corretamente.

Podemos destacar esse dígito 4 não muito bonito, que foi corretamente identificado pela rede.

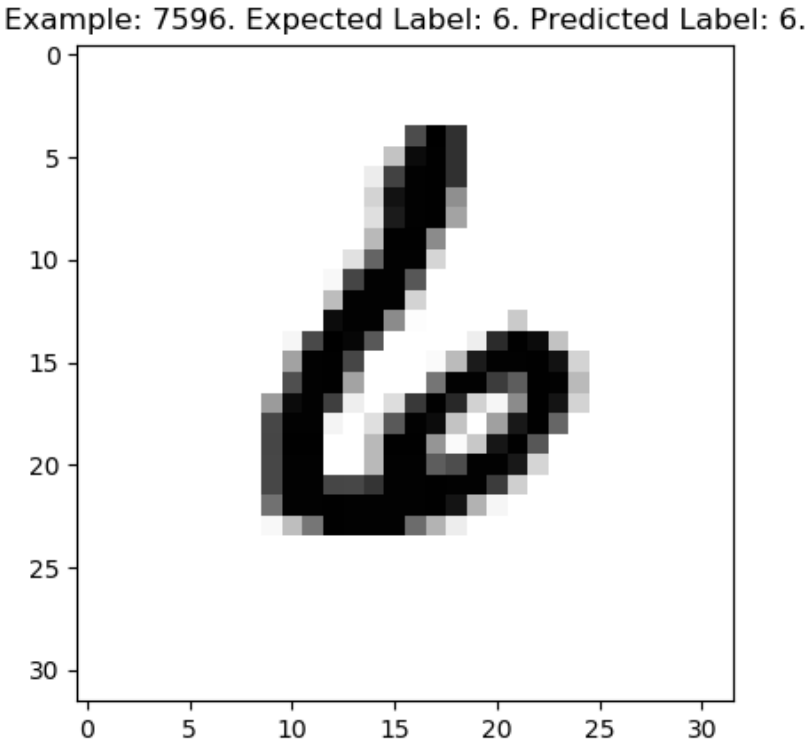

Figura 14: Exemplo de dígito que a rede identificou corretamente.