class CNN(nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(3, 6, kernel\_size=5)

        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(6, 16, kernel\_size=5)

        self.fc1 = nn.Linear(16 \* 5 \* 5, 120)

        self.fc2 = nn.Linear(120, 84)

        self.fc3 = nn.Linear(84, 2)

Epoch [10/100], Loss: 0.0711, Accuracy: 0.9680

Epoch [20/100], Loss: 0.0331, Accuracy: 0.9760

Epoch [30/100], Loss: 0.0292, Accuracy: 0.9830

Epoch [40/100], Loss: 0.0282, Accuracy: 0.9790

Epoch [50/100], Loss: 0.0186, Accuracy: 0.9860

Epoch [60/100], Loss: 0.0174, Accuracy: 0.9880

Epoch [70/100], Loss: 0.0215, Accuracy: 0.9870

Epoch [80/100], Loss: 0.0151, Accuracy: 0.9910

Epoch [90/100], Loss: 0.0169, Accuracy: 0.9930

Epoch [100/100], Loss: 0.0130, Accuracy: 0.9870

class CNN(nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(3, 6, kernel\_size=5)

        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(6, 16, kernel\_size=5)

        self.fc1 = nn.Linear(16 \* 5 \* 5, 120)

        self.drop1 = nn.Dropout(p=0.2)

        self.fc2 = nn.Linear(120, 84)

        self.fc3 = nn.Linear(84, 2)

Epoch [10/100], Loss: 0.0575, Accuracy: 0.9760

Epoch [20/100], Loss: 0.0347, Accuracy: 0.9880

Epoch [30/100], Loss: 0.0282, Accuracy: 0.9870

Epoch [40/100], Loss: 0.0318, Accuracy: 0.9880

Epoch [50/100], Loss: 0.0186, Accuracy: 0.9900

Epoch [60/100], Loss: 0.0170, Accuracy: 0.9890

Epoch [70/100], Loss: 0.0162, Accuracy: 0.9900

Epoch [80/100], Loss: 0.0182, Accuracy: 0.9910

Epoch [90/100], Loss: 0.0167, Accuracy: 0.9900

Epoch [100/100], Loss: 0.0149, Accuracy: 0.9910

class CNN(nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(3, 6, kernel\_size=5)

        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(6, 16, kernel\_size=5)

        self.fc1 = nn.Linear(16 \* 5 \* 5, 120)

        self.drop1 = nn.Dropout(p=0.2)

        self.fc2 = nn.Linear(120, 84)

        self.drop2 = nn.Dropout(p=0.2)

        self.fc3 = nn.Linear(84, 2)

Epoch [10/100], Loss: 0.0627, Accuracy: 0.9830

Epoch [20/100], Loss: 0.0407, Accuracy: 0.9890

Epoch [30/100], Loss: 0.0319, Accuracy: 0.9880

Epoch [40/100], Loss: 0.0222, Accuracy: 0.9900

Epoch [50/100], Loss: 0.0227, Accuracy: 0.9870

Epoch [60/100], Loss: 0.0184, Accuracy: 0.9880

Epoch [70/100], Loss: 0.0171, Accuracy: 0.9910

Epoch [80/100], Loss: 0.0165, Accuracy: 0.9860

Epoch [90/100], Loss: 0.0126, Accuracy: 0.9900

Epoch [100/100], Loss: 0.0239, Accuracy: 0.9920

Laten we kijken naar de testresultaten en zien hoe de toevoeging van dropout de prestaties heeft beïnvloed.

Model zonder dropout:

• Na 100 epochs: Loss = 0.0130, Accuracy = 0.9870

Dit model heeft geen dropout-lagen. Het heeft een hoge nauwkeurigheid van 0.9870 op de testgegevens. Dit betekent dat het model in staat is om correcte voorspellingen te doen voor ongeveer 98,7% van de testvoorbeelden. De lage loss-waarde geeft aan dat het model weinig fouten maakt bij het voorspellen van de klassen.

Model met dropout op één laag:

• Na 100 epochs: Loss = 0.0149, Accuracy = 0.9910

Dit model heeft een dropout-laag toegevoegd tussen de eerste lineaire laag (fc1) en de tweede lineaire laag (fc2). Het heeft een iets hogere nauwkeurigheid van 0.9910 vergeleken met het model zonder dropout. Dit betekent dat het model in staat is om correcte voorspellingen te doen voor ongeveer 99,1% van de testvoorbeelden. De loss-waarde is iets hoger, wat aangeeft dat er iets meer fouten zijn, maar over het algemeen heeft het model een betere prestatie met dropout.

Model met dropout op twee lagen:

• Na 100 epochs: Loss = 0.0239, Accuracy = 0.9920

class CNN(nn.Module):

    def \_\_init\_\_(self):

        super(CNN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(3, 6, kernel\_size=5)

        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(6, 16, kernel\_size=5)

        self.fc1 = nn.Linear(16 \* 5 \* 5, 120)

        self.drop1 = nn.Dropout(p=0.9)

        self.fc2 = nn.Linear(120, 84)

        self.drop2 = nn.Dropout(p=0.9)

        self.fc3 = nn.Linear(84, 2)

Epoch [10/100], Loss: 0.0523, Accuracy: 0.9730

Epoch [20/100], Loss: 0.0383, Accuracy: 0.9850

Epoch [30/100], Loss: 0.0341, Accuracy: 0.9850

Epoch [40/100], Loss: 0.0257, Accuracy: 0.9860

Epoch [50/100], Loss: 0.0239, Accuracy: 0.9860

Epoch [60/100], Loss: 0.0239, Accuracy: 0.9860

Epoch [70/100], Loss: 0.0158, Accuracy: 0.9890

Epoch [80/100], Loss: 0.0176, Accuracy: 0.9880

Epoch [90/100], Loss: 0.0151, Accuracy: 0.9880

Epoch [100/100], Loss: 0.0104, Accuracy: 0.9880

Dit kan te wijten zijn aan het feit dat het model te veel eenheden uitschakelt met de hoge dropout-kans, waardoor het moeilijker wordt om belangrijke patronen in de gegevens vast te leggen.

In dit specifieke geval kan een dropout-kans van 0.9 leiden tot een aanzienlijk verlies van informatie en het vermogen van het model om te leren beperken. Het is mogelijk dat het model te veel eenheden uitschakelt en hierdoor belangrijke representaties van de gegevens verliest.

<resNet zonder dropout>

* First

Epoch [30/100], Loss: 0.1353, Accuracy: 0.9620

Epoch [40/100], Loss: 0.1380, Accuracy: 0.9560

Epoch [50/100], Loss: 0.1382, Accuracy: 0.9440

Epoch [60/100], Loss: 0.1362, Accuracy: 0.9600

Epoch [70/100], Loss: 0.1434, Accuracy: 0.9630

Epoch [80/100], Loss: 0.1460, Accuracy: 0.9650

Epoch [90/100], Loss: 0.1350, Accuracy: 0.9580

Epoch [100/100], Loss: 0.1485, Accuracy: 0.9610

* Second

Epoch [10/100], Loss: 0.1507, Accuracy: 0.9430

Epoch [20/100], Loss: 0.1711, Accuracy: 0.9370

Epoch [30/100], Loss: 0.1447, Accuracy: 0.9330

Epoch [40/100], Loss: 0.1275, Accuracy: 0.9300

Epoch [50/100], Loss: 0.1310, Accuracy: 0.9300

Epoch [60/100], Loss: 0.1321, Accuracy: 0.9490

Epoch [70/100], Loss: 0.1297, Accuracy: 0.9480

Epoch [80/100], Loss: 0.1291, Accuracy: 0.9460

Epoch [90/100], Loss: 0.1333, Accuracy: 0.9410

Epoch [100/100], Loss: 0.1462, Accuracy: 0.9410

< Model Fit & Model Freeze>

poch [10/100], Loss: 0.6706, Accuracy: 0.9420

Epoch [20/100], Loss: 0.6477, Accuracy: 0.9340

Epoch [30/100], Loss: 0.6264, Accuracy: 0.9290

Epoch [40/100], Loss: 0.6067, Accuracy: 0.9260

Epoch [50/100], Loss: 0.5884, Accuracy: 0.9240

Epoch [60/100], Loss: 0.5714, Accuracy: 0.9230

Epoch [70/100], Loss: 0.5556, Accuracy: 0.9210

Epoch [80/100], Loss: 0.5409, Accuracy: 0.9230

Epoch [90/100], Loss: 0.5271, Accuracy: 0.9220

Epoch [100/100], Loss: 0.5143, Accuracy: 0.9220