Deep Learning - Assignment 3

Semi Supervised Learning com o algorítmo FixMatch.

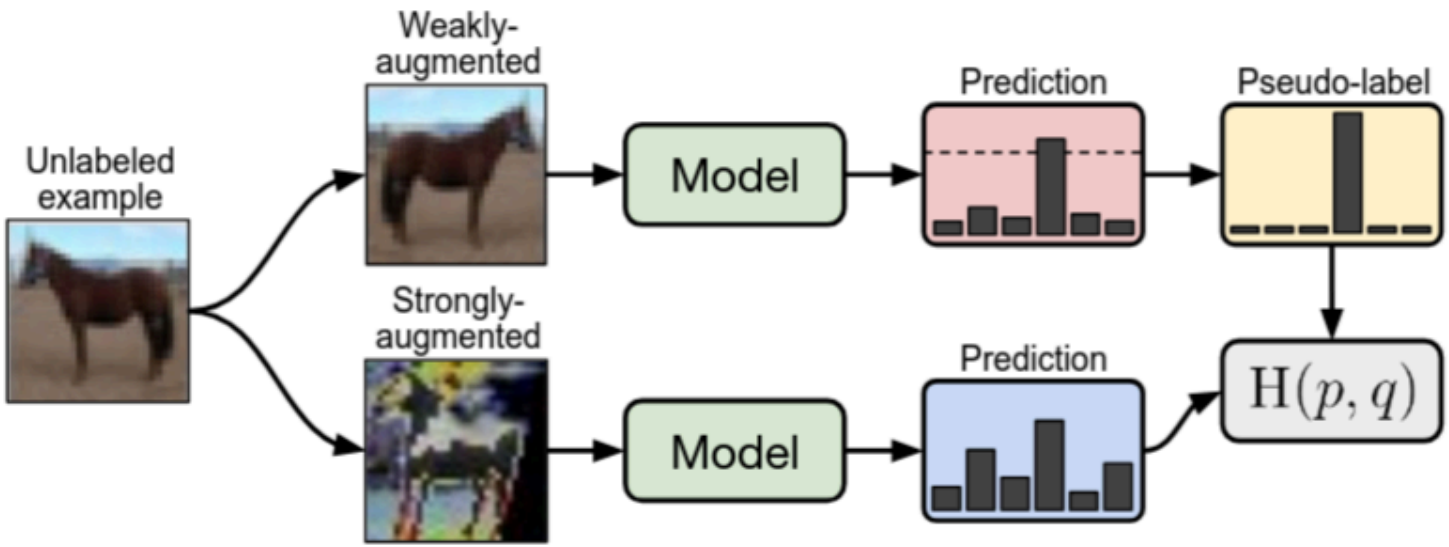
Guilherme Castilho, Pedro Tokar e Vitor do Nascimento

Fluxo de trabalho

Fluxo de trabalho

- Plataformas e ferramentas utilizadas: Google Colab, Kaggle e VS Code;
- Criação das funções necessárias para a experimentação;
- Experimentação com diferentes números de samples por classe;
- Experimentação com cut-out;
- Experimentação com amostras representativas e não representativas;

Algoritmo


Unlabeled example
Weakly-augmented
Strongly-augmented
Model
Model
Prediction
Prediction
Pseudo-label
H(p, q)

Fluxo de trabalho

Todos os testes foram executados com os hiperparâmetros e schedulers listados no paper:

- Batch size 64;
- Learning rate: 0.003;
- Weight decay: 0.0005;
- Parâmetros da Loss: $\tau = 0.95, \lambda_u = 1$;
- Proporção do batch sem labels: $\mu = 7$;
- Scheduler com função $\cos(\frac{7\pi k}{16K})$;

Implementação

Dataset


```
#Labeled dataset
def __getitem__(self, idx):
      real_idx = self.indexes[idx]
      image, label = self.dataset[real_idx]
      image = image.to(torch.float).to(self.device)

      image = self.weak_augmentations(image)
      if NORMALIZE:
          image = self.normalize_img(image)

      return image, label

#Unlabeled dataset
def __getitem__(self, idx):
      real_idx = self.indexes[idx]
      image, label = self.dataset[real_idx]
      image = image.to(torch.uint8).to(self.device)

      weak_image = self.weak_augmentations(image).to(torch.float)
      strong_image = self.strong_augmentations(image).to(torch.float)
      if NORMALIZE:
          weak_image = self.normalize_img(weak_image)
          strong_image = self.normalize_img(strong_image)

      return weak_image, strong_image
```

Dataset


```python
def get_split_dataset(dataset, n_classes, n_samples, ctaugment = True, device = "cpu"):
  labeled_indexes = []

  random_indexes = torch.randperm(len(dataset))

  total_indexes = n_classes * n_samples
  frequencies = defaultdict(lambda: 0)
  curr = 0

  while len(labeled_indexes) < total_indexes:
      if curr == len(dataset):
          raise RuntimeError("Não foi possível fazer split do dataset")

      label = dataset[random_indexes[curr]][1]
      if frequencies[label] < n_samples:
          labeled_indexes.append(random_indexes[curr].item())
          frequencies[label] += 1
      curr += 1

  unlabeled_indexes = list(set(range(len(dataset))) - set(labeled_indexes))

  labeled_dataset = LabeledDataset(dataset, labeled_indexes, device)
  unlabeled_dataset = UnlabeledDataset(dataset, unlabeled_indexes, ctaugment, device)
  return labeled_dataset, unlabeled_dataset
```

Loss


```
class FixMatchLoss(nn.Module):
    def __init__(self, threshold, weight):
        super().__init__()
        self.threshold = threshold
        self.weight = weight
        self.cross_entropy = nn.CrossEntropyLoss(reduction = "sum")
        self.softmax = nn.Softmax(dim = 1)

    def forward(self, labeled_predictions, labeled_truth, unlabeled_weak_predictions, unlabeled_strong_predictions):
        l_s = self.cross_entropy(labeled_predictions, labeled_truth)/labeled_truth.shape[0]

        with torch.no_grad():
            unlabeled_weak_predictions = self.softmax(unlabeled_weak_predictions)
            mask = unlabeled_weak_predictions.max(dim = 1)[0] > self.threshold
            if sum(mask) == 0:
                return l_s, 0

        l_u = self.cross_entropy(
            unlabeled_strong_predictions[mask],
            unlabeled_weak_predictions[mask].argmax(dim = 1)
        )/unlabeled_weak_predictions.shape[0]

        return l_s + self.weight * l_u, sum(mask)
```

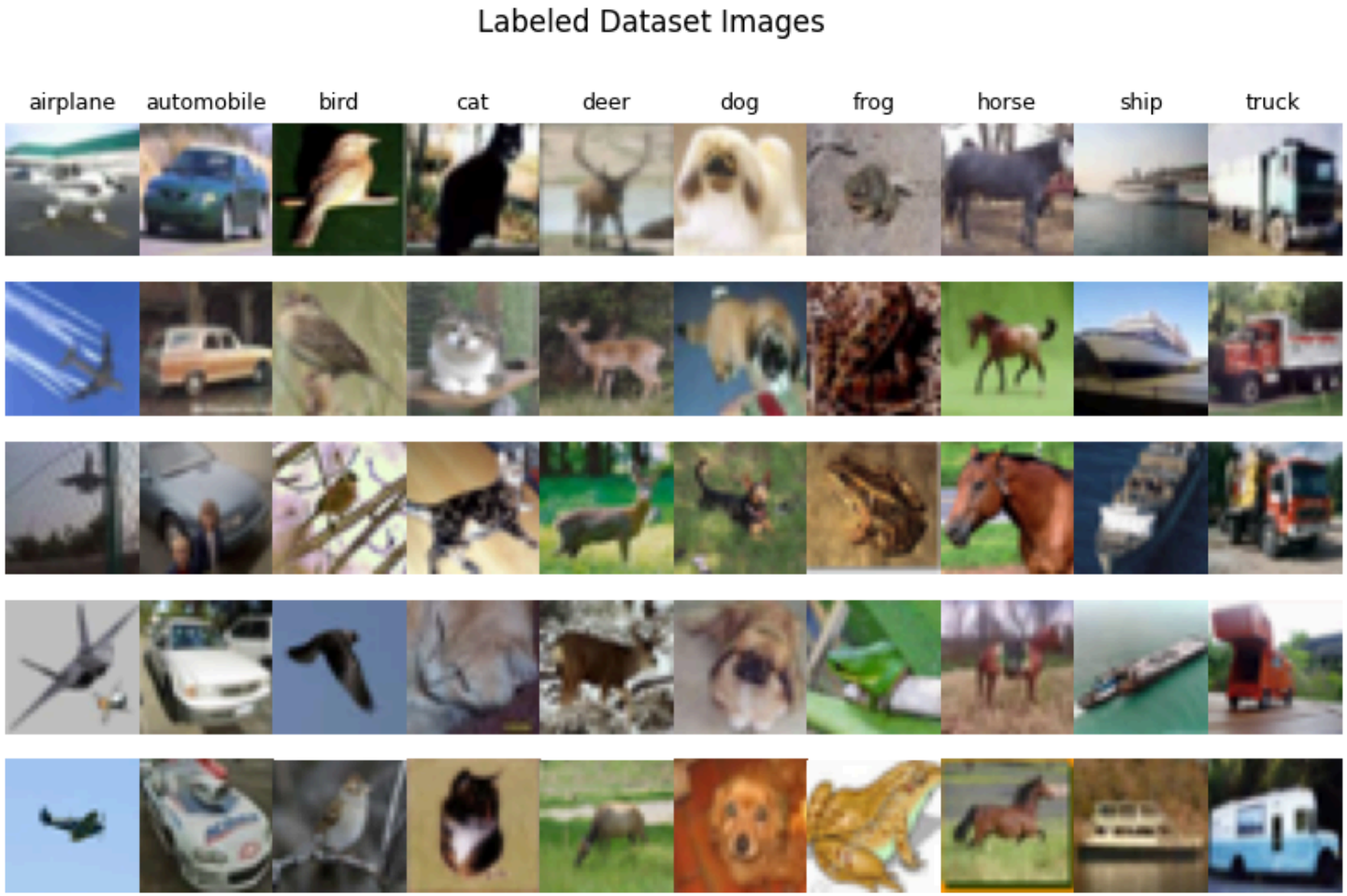
```
zip_loaders = zip(cycle(self._labeled_dataloader), self._unlabeled_dataloader)
progress_bar = tqdm(zip_loaders, desc = f"Epoch {epoch + 1} | Training", total = len(self._unlabeled_dataloader))

for i, (labeled_data, unlabeled_data) in enumerate(progress_bar):
    labeled_images, labeled_labels = labeled_data
    unlabeled_weak, unlabeled_strong = unlabeled_data
    #[...]
    self._optimizer.zero_grad()

    labeled_predictions = self._model(labeled_images)
    unlabeled_weak_predictions = self._model(unlabeled_weak)
    unlabeled_strong_predictions = self._model(unlabeled_strong)

    loss, confident = self._criterion(labeled_predictions,
                                      labeled_labels,
                                      unlabeled_weak_predictions,
                                      unlabeled_strong_predictions)
    loss.backward()
    self._optimizer.step()

    running_loss += loss.item()
    progress_bar.set_postfix(loss = loss.item(), lr = self._optimizer.param_groups[0]["lr"])
    confidents += int(confident)

epoch_loss = running_loss / len(self._unlabeled_dataloader)
percentage_confiance = 100*confidents/len(self._unlabeled_dataloader.dataset)
```

Treino


```
optimizer = optim.SGD(
    model.parameters(),
    lr = learning_rate,
    momentum = 0.9,
    nesterov = True,
    weight_decay = 0.0005
)
scheduler = optim.lr_scheduler.LambdaLR(
    optimizer,
    lambda epoch: cos((7*pi*epoch) / (16*epochs))
)
loss_function = FixMatchLoss(threshold, lambda_u)

trainer = Trainer(
    model,
    labeled_dataset,
    unlabeled_dataset,
    loss_function,
    optimizer,
    val_dataset = test_dataset,
    device = DEVICE
)
```

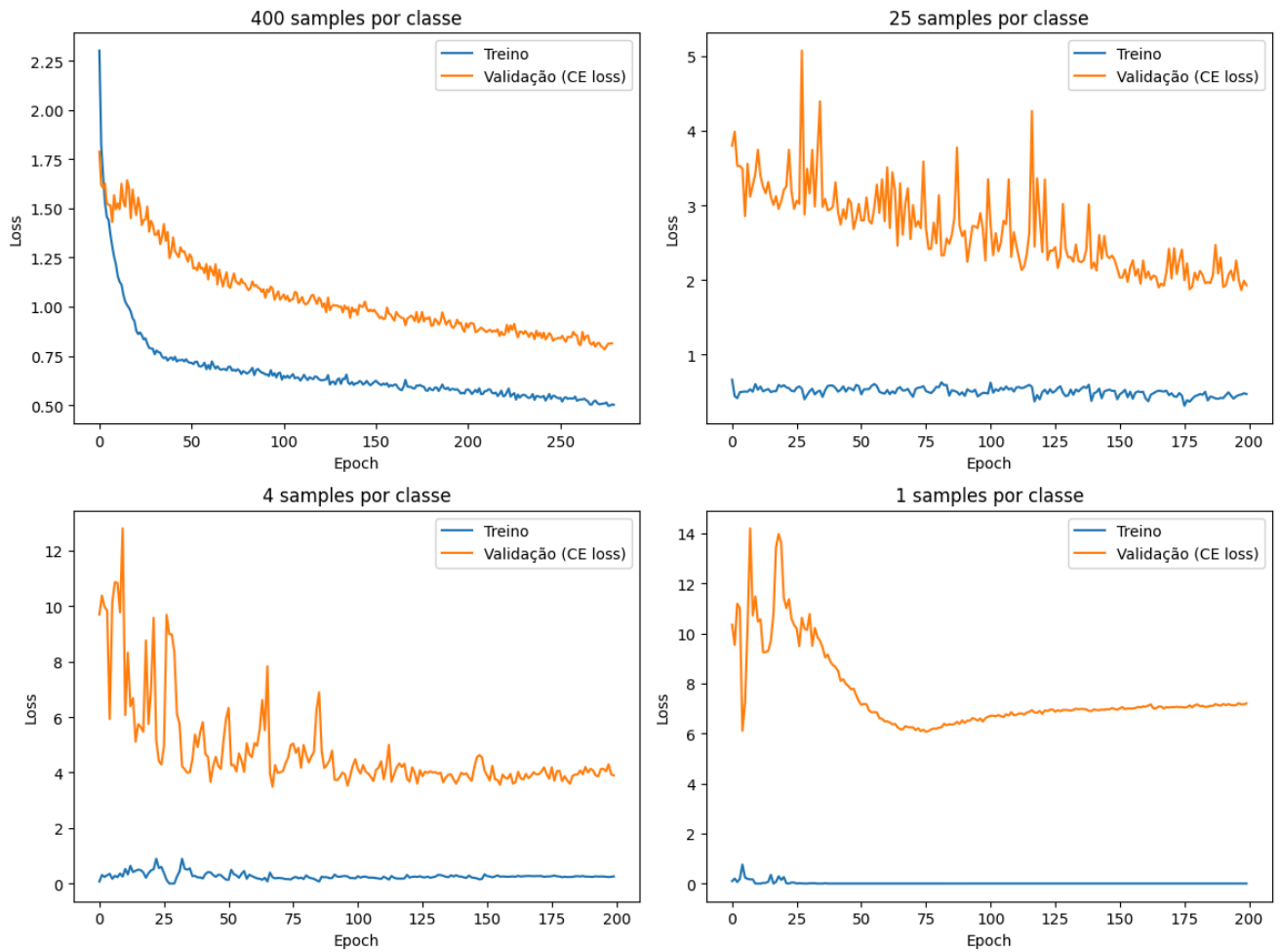
Imagens

Imagens

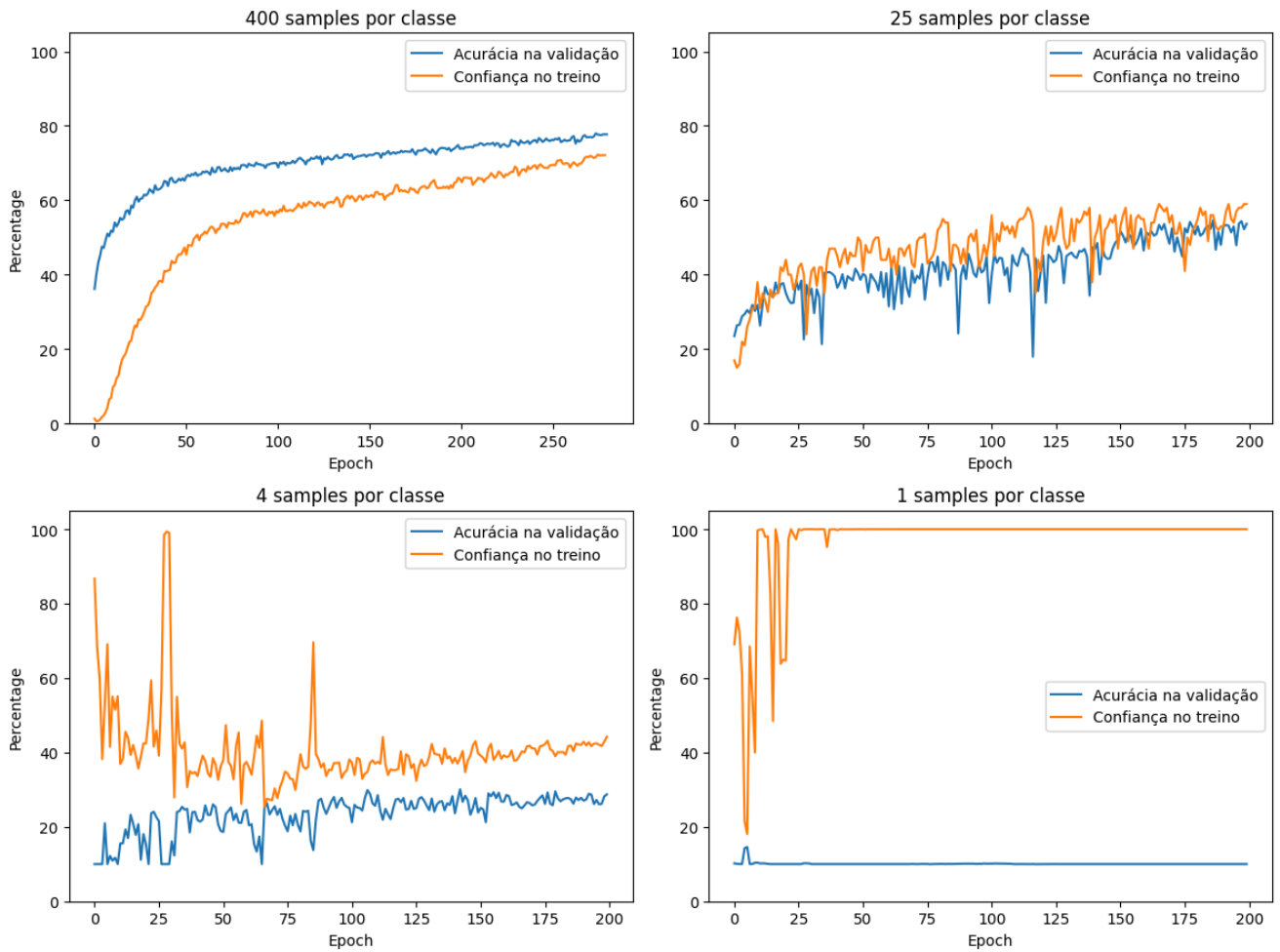

Labeled Dataset Images
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Experimento: número de samples

Loss


400 samples por classe
Treino
Validação (CE loss)
Loss
Epoch
25 samples por classe
Treino
Validação (CE loss)
Loss
Epoch
4 samples por classe
Treino
Validação (CE loss)
Loss
Epoch
1 samples por classe
Treino
Validação (CE loss)
Loss
Epoch

Accuracy


400 samples por classe
25 samples por classe
4 samples por classe
1 samples por classe
Acurácia na validação
Confiança no treino
Percentage
Epoch

Augmentation

AutoAugment, RandAugment e CTAugment

- AutoAugment é um método para aprender uma política de aumento de dados que resulta em alta acurácia no conjunto de validação;
- Política de aumento: sequência de tuplas (transformação, magnitude do parâmetro);
- No AutoAugment a politica é aprendida com supervisão;
- RandAugment: Resolve a questão do treinamento supervisionado por meio de amostragem aleatória das transformações, mas exige ajuste nos hiperparâmetros;
- CTAugment: Assim como o RandAugment, também amostra transformações uniformemente ao acaso, mas infere dinamicamente as magnitudes para cada transformação durante o processo de treinamento.

Augmentation

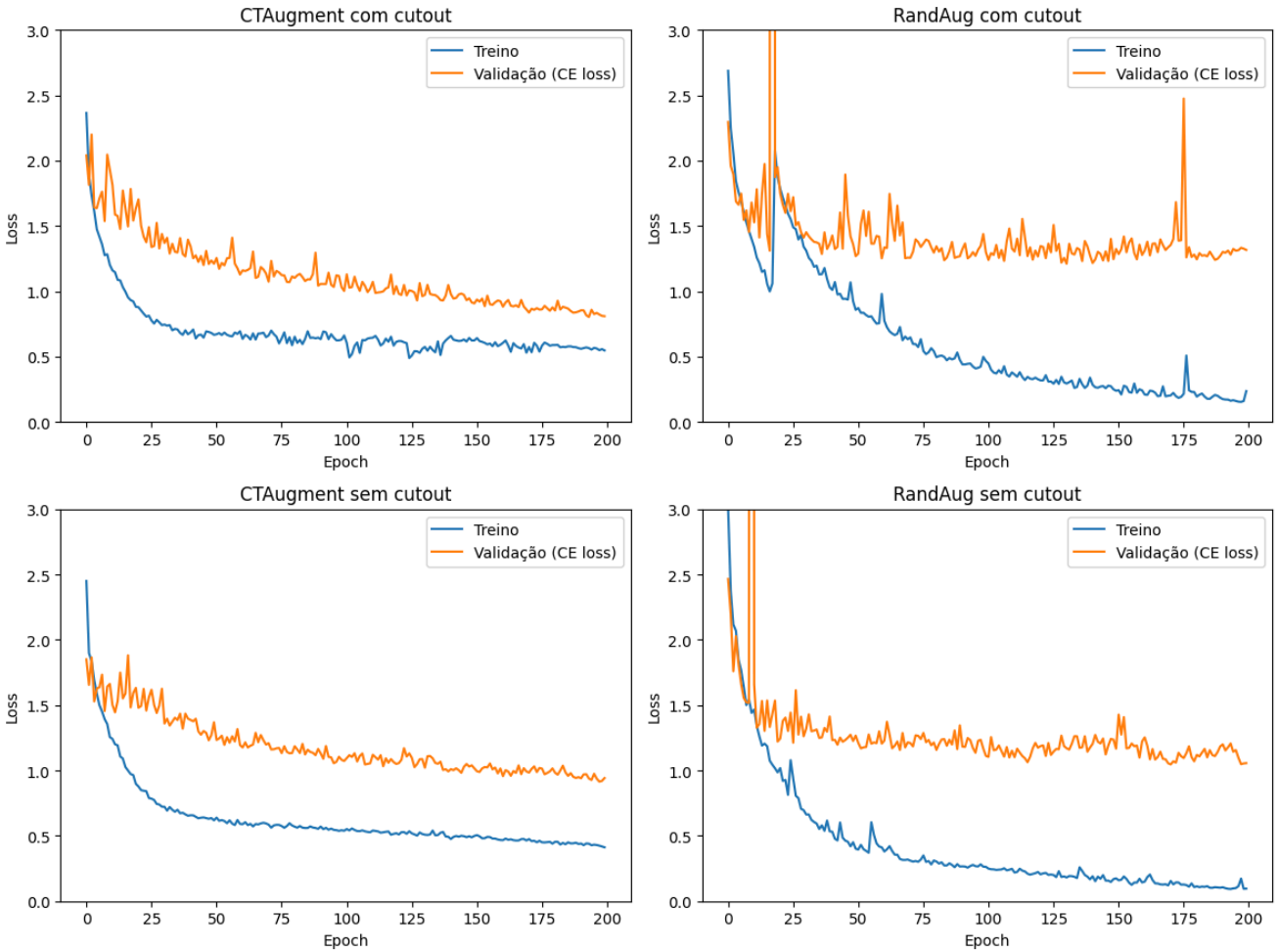
Transformation	Description	Parameter	Range
Autocontrast	Maximizes the image contrast by setting the darkest (lightest) pixel to black (white).		
Brightness	Adjusts the brightness of the image. $B = 0$ returns a black image, $B = 1$ returns the original image.	B	[0.05, 0.95]
Color	Adjusts the color balance of the image like in a TV. $C = 0$ returns a black & white image, $C = 1$ returns the original image.	C	[0.05, 0.95]
Contrast	Controls the contrast of the image. A $C = 0$ returns a gray image, $C = 1$ returns the original image.	C	[0.05, 0.95]
Equalize	Equalizes the image histogram.		
Identity	Returns the original image.		
Posterize	Reduces each pixel to B bits.	B	[4, 8]
Rotate	Rotates the image by θ degrees.	θ	[-30, 30]
Sharpness	Adjusts the sharpness of the image, where $S = 0$ returns a blurred image, and $S = 1$ returns the original image.	S	[0.05, 0.95]
Shear_x	Shears the image along the horizontal axis with rate R.	R	[-0.3, 0.3]
Shear_y	Shears the image along the vertical axis with rate R.	R	[-0.3, 0.3]
Solarize	Inverts all pixels above a threshold value of T.	T	[0, 1]
Translate_x	Translates the image horizontally by ($\lambda \times$image width) pixels.	λ	[-0.3, 0.3]
Translate_y	Translates the image vertically by ($\lambda \times$image height) pixels.	λ	[-0.3, 0.3]

Augmentation

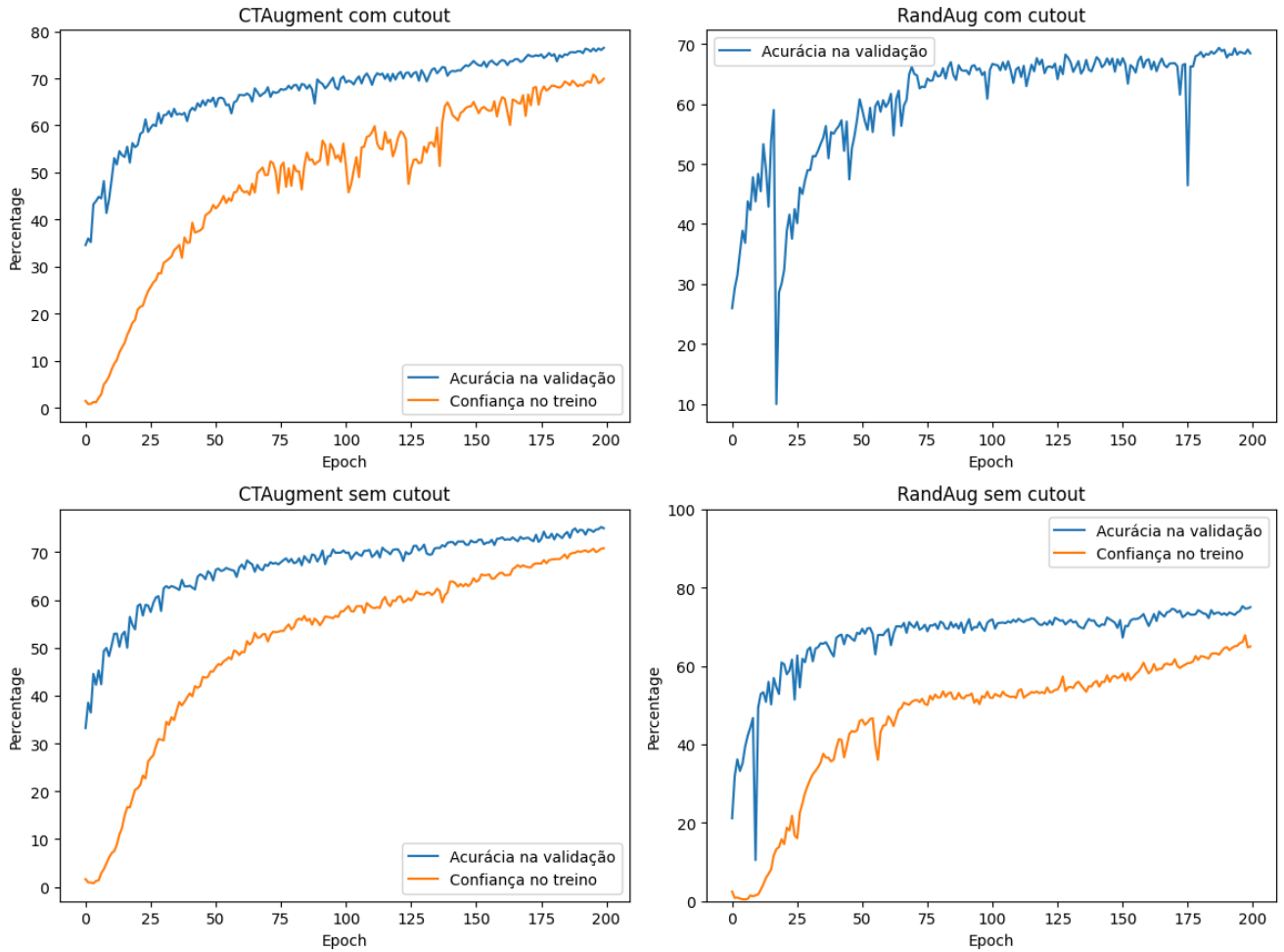
Transformation	Description	Parameter	Range
Autocontrast	Maximizes the image contrast by setting the darkest (lightest) pixel to black (white), and then blends with the original image with blending ratio λ.	λ	[0, 1]
Brightness	Adjusts the brightness of the image. $B = 0$ returns a black image, $B = 1$ returns the original image.	B	[0, 1]
Color	Adjusts the color balance of the image like in a TV. $C = 0$ returns a black & white image, $C = 1$ returns the original image.	C	[0, 1]
Contrast	Controls the contrast of the image. A $C = 0$ returns a gray image, $C = 1$ returns the original image.	C	[0, 1]
Cutout	Sets a random square patch of side-length ($L\times$image width) pixels to gray.	L	[0, 0.5]
Equalize	Equalizes the image histogram, and then blends with the original image with blending ratio λ.	λ	[0, 1]
Invert	Inverts the pixels of the image, and then blends with the original image with blending ratio λ.	λ	[0, 1]
Identity	Returns the original image.		
Posterize	Reduces each pixel to B bits.	B	[1, 8]
Rescale	Takes a center crop that is of side-length ($L\times$image width), and rescales to the original image size using method M.	L	[0.5, 1.0]
		M	see caption
Rotate	Rotates the image by θ degrees.	θ	[-45, 45]
Sharpness	Adjusts the sharpness of the image, where $S = 0$ returns a blurred image, and $S = 1$ returns the original image.	S	[0, 1]
Shear_x	Shears the image along the horizontal axis with rate R.	R	[-0.3, 0.3]
Shear_y	Shears the image along the vertical axis with rate R.	R	[-0.3, 0.3]
Smooth	Adjusts the smoothness of the image, where $S = 0$ returns a maximally smooth image, and $S = 1$ returns the original image.	S	[0, 1]
Solarize	Inverts all pixels above a threshold value of T.	T	[0, 1]
Translate_x	Translates the image horizontally by ($\lambda\times$image width) pixels.	λ	[-0.3, 0.3]
Translate_y	Translates the image vertically by ($\lambda\times$image height) pixels.	λ	[-0.3, 0.3]

Experimento: Augmentation e Cutout

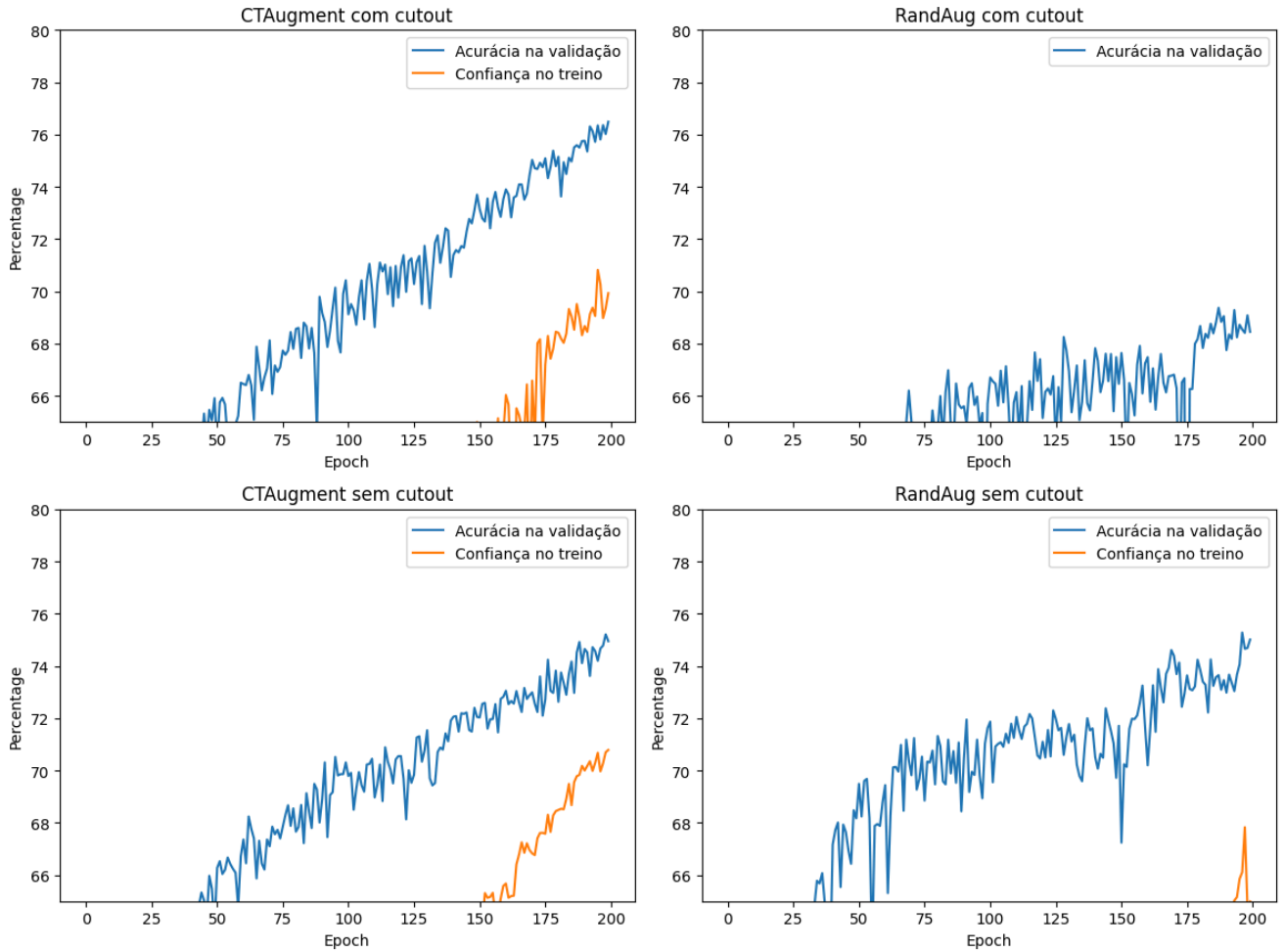
Experimento: Augmentation e Cutout



Experimento: Augmentation e Cutout



Experimento: Augmentation e Cutout



Experimento:
Imagens pouco representativas

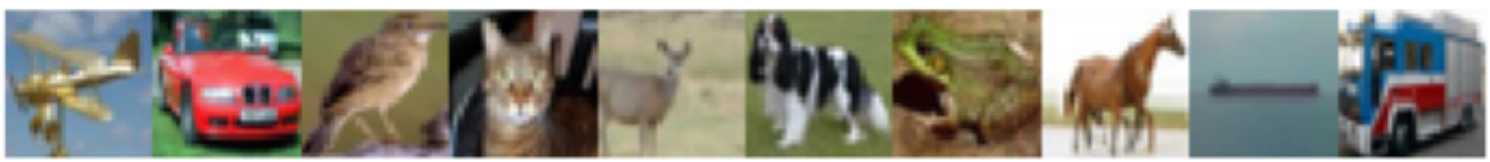
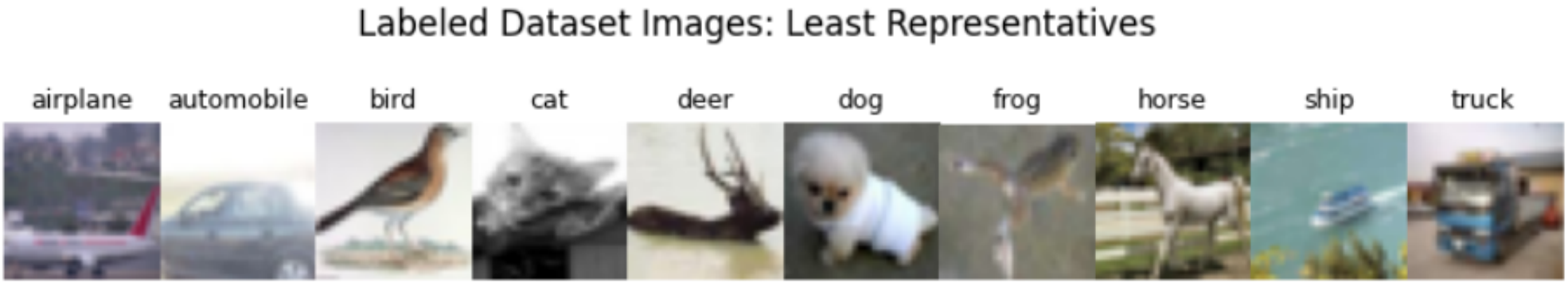


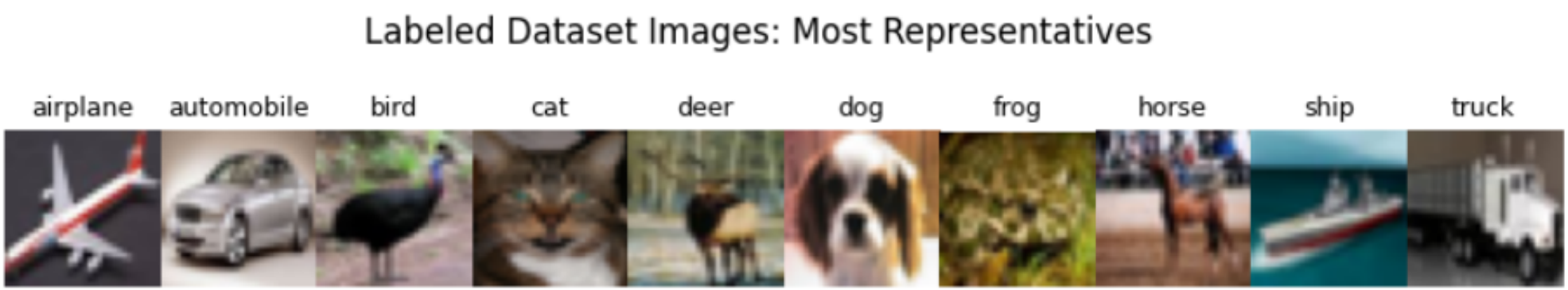
Figure 2: FixMatch reaches 78% CIFAR-10 accuracy using only above 10 labeled images.

Definição do Modelo

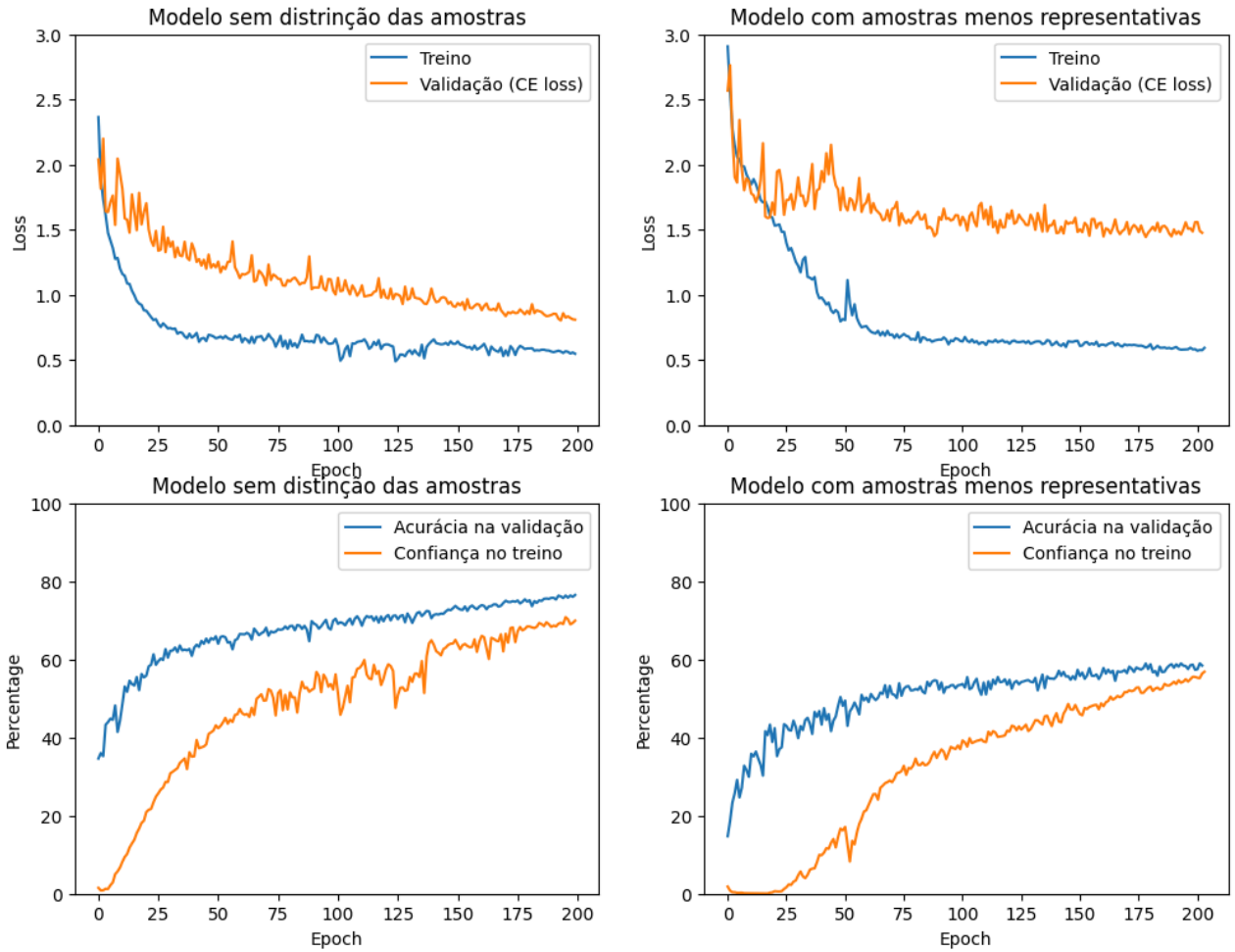

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 64, 3, padding=1)

        self.fc1 = nn.Linear(64 * 8 * 8, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = F.relu(self.conv3(x))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Verificando Influencia da representatividade





Resultado


Modelo sem distrinção das amostras
Treino
Validação (CE loss)
Loss
Epoch
Modelo com amostras menos representativas
Treino
Validação (CE loss)
Loss
Epoch
Modelo sem distinção das amostras
Acurácia na validação
Confiança no treino
Percentage
Epoch
Modelo com amostras menos representativas
Acurácia na validação
Confiança no treino
Percentage
Epoch