UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FABRÍCIO MEDEIROS TOZO

CLASSIFICAÇÃO DE IMAGENS COM REDE NEURAL CONVOLUCIONAL

ALEGRE
2024

FABRÍCIO MEDEIROS TOZO

CLASSIFICAÇÃO DE IMAGENS COM REDE NEURAL CONVOLUCIONAL

Este documento tem o objetivo de relatar as etapas de desenvolvimento do primeiro trabalho da disciplina de Aprendizado de Máquina.
Professor: Jacson Rodrigues Correia da Silva

ALEGRE
2024

SUMÁRIO

1 INTRODUÇÃO

Este relatório tem como objetivo descrever as etapas e resultados obtidos durante o desenvolvimento de um classificador de imagens utilizando redes neurais convolucionais.

O trabalho foi realizado utilizando o dataset Flowers102, que contém 102 categorias diferentes de flores.

O foco foi selecionar um subconjunto de 50 classes e aplicar um modelo pré-treinado ResNet18 para realizar a classificação das imagens, utilizando técnicas de Cross-Validation e ajuste de hiperparâmetros.

Além disso, serão descritas as dificuldades encontradas, os resultados obtidos, e uma análise crítica sobre o aprendizado adquirido ao longo do processo.

Palavras-chave: Aprendizado supervisionado; Classificação; Aprendizado Profundo; Redes Neurais Artificiais;

2 OBJETIVOS

2.0.1 OBJETIVOS ESPECÍFICOS

- Selecionar, aleatoriamente, 50 classes do dataset Flowers102;
- Pré processamento dos dados;
- Seleção de hyperparametros;
- Fase de Treino, Teste e Validação;
- Analise da acurácia e curva de aprendizado;
- Descrição do aprendizado com o trabalho;
- Referencias;

3 50 CLASSES DO DATASET FLOWERS102

A etapa da separação dos dados consiste majoritariamente em escolher as imagens a serem usadas como entrada para o treinamento.

Usamos nessa pesquisa as seguintes 50 classes aleatórias.

```
def filter_classes(dataset, selected_classes):
    indices = [idx for idx, (_, label) in enumerate(dataset)
               if label in selected_classes]
    return Subset(dataset, indices)

    [ 7, 18, 64, 14, 8, 47, 94, 84, 30, 74, 27, 70, 33, 1, 28, 95, 32,
    72, 77, 60, 55, 79, 11, 54, 59, 29, 51, 43, 71,  5, 92, 53, 34, 2,
    48, 36, 9, 39, 42, 15, 17, 86, 45, 93, 62, 49, 46, 82, 68, 21]
```

4 PRÉ PROCESSAMENTO DOS DADOS E SELEÇÃO DE HYPERPARAMETROS

Com as Classes selecionadas adicionamos uma nova classe para remapear os indices.

```
class MappedDataset(Dataset):
    def __init__(self, subset, class_mapping):
        self.subset = subset
        self.class_mapping = class_mapping

    def __len__(self):
        return len(self.subset)

    def __getitem__(self, idx):
        img, label = self.subset[idx]
        label = self.class_mapping.get(label, -1)
        return img, label
```

```
def prepare_data(self, data_transform, train_dataset, val_dataset,
   test_dataset, selected_classes):
    # Mapear classes para [0, 49]
    self.class_mapping = {original: new for new, original in enumerate(
       selected_classes)}

    # Filtragem e mapeamento dos datasets
    train_subset = self.filter_classes(train_dataset, selected_classes)
    val_subset = self.filter_classes(val_dataset, selected_classes)
    test_subset = self.filter_classes(test_dataset, selected_classes)

    train_mapped = self.MappedDataset(train_subset, self.class_mapping)
    val_mapped = self.MappedDataset(val_subset, self.class_mapping)
    test_mapped = self.MappedDataset(test_subset, self.class_mapping)

    # DataLoaders
    self.train_loader = DataLoader(train_mapped, batch_size=self.
       batch_size, shuffle=True)
    self.val_loader = DataLoader(val_mapped, batch_size=self.batch_size,
        shuffle=False)
    self.test_loader = DataLoader(test_mapped, batch_size=self.
       batch_size, shuffle=False)
```

5 SELEÇÃO DE HYPERPARAMETROS E DATA AUGMENTATIONS

```
    def create_model(self):
        model = models.resnet18()
        num_ftrs = model.fc.in_features
        model.fc = nn.Sequential(
            nn.Linear(num_ftrs, self.num_classes)
        )
        return model.to(device)
```

Data Augmentations utilizados

```
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.3, contrast=0.3),
    transforms.RandomAffine(degrees=30, shear=20),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
       0.225])
])

train_dataset = datasets.Flowers102(root=r'E:\Dataset_torchvision',
   split='test', download=True, transform=data_transform)
val_dataset = datasets.Flowers102(root=r'E:\Dataset_torchvision', split=
   'val', download=True, transform=data_transform)
test_dataset = datasets.Flowers102(root=r'E:\Dataset_torchvision', split
   ='train', download=True, transform=data_transform)
```

Melhores Hyperparametros encontrados

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam()
num_classes=50
batch_size=32
 lr= 0.001 ~~ lr= 0.0001
epochs=21
```

Insights: Learning Rate = 0.001 apresentou melhor resultado nos primeiros 2 treinos, entrando na terceira fase de treino temos melhor resultados usando Learning Rate scheduler entre 0.001 e 0.00001.

6 FASE DE TREINO, TESTE E VALIDAÇÃO

Nesta fase já com todas as imagens selecionadas, mascaras criadas e organizadas podemos entrar na fase de treino e teste.

```
def train(self):
    print("Starting␣training...")
    train_losses = []
    val_accuracies = []
    val_f1_scores = []

    for epoch in range(self.epochs):
        self.model.train()
        running_loss = 0.0
        for inputs, labels in self.train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.criterion(outputs, labels)
            loss.backward()
            self.optimizer.step()
            running_loss += loss.item() * inputs.size(0)

        epoch_loss = running_loss / len(self.train_loader.dataset)
        train_losses.append(epoch_loss)
        print(f'Epoch{epoch␣+␣1}/{self.epochs},␣Loss:␣{epoch_loss:.4f}')

        #Validacao apos cada epoca
        val_accuracy, val_f1 = self.validate(self.val_loader)
        val_accuracies.append(val_accuracy)
        val_f1_scores.append(val_f1)

    print("Training␣complete.")
    return train_losses, val_accuracies, val_f1_scores
```

```
def validate(self, loader):
    self.model.eval()
    correct = 0
    total = 0
    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = self.model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = accuracy_score(all_labels, all_predictions) * 100
    f1 = f1_score(all_labels, all_predictions, average='macro')
    print(f'Validation Accuracy: {accuracy:.2f}%, Macro F1-Score: {f1:.4
        f}')
    return accuracy, f1

# Funcao para testar o modelo com o conjunto de teste
def test(self):
    print("Testing model on test dataset...")
    return self.validate(self.test_loader)
```

7 ANALISE DA ACURÁCIA E CURVA DE APRENDIZADO

```
Using device: cuda
Starting training...
Epoch 1/21, Loss: 3.0818
Validation Accuracy: 17.80%, Macro F1-Score: 0.1153
Epoch 2/21, Loss: 2.2746
Validation Accuracy: 32.60%, Macro F1-Score: 0.2693
Epoch 3/21, Loss: 1.9799
Validation Accuracy: 40.80%, Macro F1-Score: 0.3722
Epoch 4/21, Loss: 1.7698
Validation Accuracy: 43.00%, Macro F1-Score: 0.3906
Epoch 5/21, Loss: 1.5682
Validation Accuracy: 45.40%, Macro F1-Score: 0.4136
Epoch 6/21, Loss: 1.4402
Validation Accuracy: 52.00%, Macro F1-Score: 0.4902
Epoch 7/21, Loss: 1.3238
Validation Accuracy: 56.00%, Macro F1-Score: 0.5307
Epoch 8/21, Loss: 1.1942
Validation Accuracy: 52.00%, Macro F1-Score: 0.5013
Epoch 9/21, Loss: 1.1342
Validation Accuracy: 58.80%, Macro F1-Score: 0.5625
Epoch 10/21, Loss: 1.0470
Validation Accuracy: 57.80%, Macro F1-Score: 0.5557
Epoch 11/21, Loss: 0.9732
Validation Accuracy: 56.80%, Macro F1-Score: 0.5420
Epoch 12/21, Loss: 0.9217
Validation Accuracy: 62.60%, Macro F1-Score: 0.6070
Epoch 13/21, Loss: 0.8037
Validation Accuracy: 64.60%, Macro F1-Score: 0.6167
Epoch 14/21, Loss: 0.7640
Validation Accuracy: 65.80%, Macro F1-Score: 0.6385
Epoch 15/21, Loss: 0.7526
Validation Accuracy: 68.80%, Macro F1-Score: 0.6688
Epoch 16/21, Loss: 0.6704
Validation Accuracy: 71.20%, Macro F1-Score: 0.6988
Epoch 17/21, Loss: 0.6542
Validation Accuracy: 69.80%, Macro F1-Score: 0.6790
Epoch 18/21, Loss: 0.5910
Validation Accuracy: 70.40%, Macro F1-Score: 0.6884
Epoch 19/21, Loss: 0.5793
Validation Accuracy: 71.60%, Macro F1-Score: 0.7095
Epoch 20/21, Loss: 0.5633
Validation Accuracy: 66.60%, Macro F1-Score: 0.6591
Epoch 21/21, Loss: 0.4815
Validation Accuracy: 76.80%, Macro F1-Score: 0.7602
Training complete.
```

```
Using device: cuda
Starting training...
Epoch 1/21, Loss: 0.5201
Validation Accuracy: 75.00%, Macro F1-Score: 0.7482
Epoch 2/21, Loss: 0.4587
Validation Accuracy: 72.00%, Macro F1-Score: 0.7213
Epoch 3/21, Loss: 0.4303
Validation Accuracy: 71.60%, Macro F1-Score: 0.7126
Epoch 4/21, Loss: 0.4071
Validation Accuracy: 76.60%, Macro F1-Score: 0.7547
Epoch 5/21, Loss: 0.4069
Validation Accuracy: 71.60%, Macro F1-Score: 0.7076
Epoch 6/21, Loss: 0.3806
Validation Accuracy: 77.60%, Macro F1-Score: 0.7739
Epoch 7/21, Loss: 0.3589
Validation Accuracy: 78.60%, Macro F1-Score: 0.7788
Epoch 8/21, Loss: 0.3313
Validation Accuracy: 75.00%, Macro F1-Score: 0.7384
Epoch 9/21, Loss: 0.3125
Validation Accuracy: 79.00%, Macro F1-Score: 0.7858
Epoch 10/21, Loss: 0.2959
Validation Accuracy: 73.20%, Macro F1-Score: 0.7296
Epoch 11/21, Loss: 0.2763
Validation Accuracy: 79.40%, Macro F1-Score: 0.7942
Epoch 12/21, Loss: 0.2714
Validation Accuracy: 78.00%, Macro F1-Score: 0.7770
Epoch 13/21, Loss: 0.2592
Validation Accuracy: 77.20%, Macro F1-Score: 0.7721
Epoch 14/21, Loss: 0.2767
Validation Accuracy: 79.80%, Macro F1-Score: 0.7898
Epoch 15/21, Loss: 0.2232
Validation Accuracy: 81.60%, Macro F1-Score: 0.8159
Epoch 16/21, Loss: 0.2143
Validation Accuracy: 78.00%, Macro F1-Score: 0.7734
Epoch 17/21, Loss: 0.2322
Validation Accuracy: 77.80%, Macro F1-Score: 0.7756
Epoch 18/21, Loss: 0.2362
Validation Accuracy: 79.00%, Macro F1-Score: 0.7810
Epoch 19/21, Loss: 0.1946
Validation Accuracy: 80.40%, Macro F1-Score: 0.7954
Epoch 20/21, Loss: 0.1688
Validation Accuracy: 79.80%, Macro F1-Score: 0.7922
Epoch 21/21, Loss: 0.1570
Validation Accuracy: 81.00%, Macro F1-Score: 0.8042
Training complete.
```

```
Using device: cuda
Starting training...
Epoch 1/21, Loss: 0.2017
Validation Accuracy: 78.80%, Macro F1-Score: 0.7786
Epoch 2/21, Loss: 0.1699
Validation Accuracy: 80.60%, Macro F1-Score: 0.7987
Epoch 3/21, Loss: 0.1950
Validation Accuracy: 80.80%, Macro F1-Score: 0.8042
Epoch 4/21, Loss: 0.1722
Validation Accuracy: 78.00%, Macro F1-Score: 0.7662
Epoch 5/21, Loss: 0.1961
Validation Accuracy: 82.00%, Macro F1-Score: 0.8184
Epoch 6/21, Loss: 0.1617
Validation Accuracy: 83.40%, Macro F1-Score: 0.8291
Epoch 7/21, Loss: 0.1470
Validation Accuracy: 84.20%, Macro F1-Score: 0.8369
Epoch 8/21, Loss: 0.1448
Validation Accuracy: 82.80%, Macro F1-Score: 0.8249
Epoch 9/21, Loss: 0.1584
Validation Accuracy: 79.80%, Macro F1-Score: 0.7892
Epoch 10/21, Loss: 0.1263
Validation Accuracy: 84.20%, Macro F1-Score: 0.8386
Epoch 11/21, Loss: 0.1052
Validation Accuracy: 81.60%, Macro F1-Score: 0.8123
Epoch 12/21, Loss: 0.1116
Validation Accuracy: 82.20%, Macro F1-Score: 0.8149
Epoch 13/21, Loss: 0.1565
Validation Accuracy: 83.20%, Macro F1-Score: 0.8224
Epoch 14/21, Loss: 0.0906
Validation Accuracy: 83.00%, Macro F1-Score: 0.8179
Epoch 15/21, Loss: 0.0931
Validation Accuracy: 81.40%, Macro F1-Score: 0.8092
Epoch 16/21, Loss: 0.1244
Validation Accuracy: 82.60%, Macro F1-Score: 0.8233
Epoch 17/21, Loss: 0.1068
Validation Accuracy: 83.60%, Macro F1-Score: 0.8311
Epoch 18/21, Loss: 0.0935
Validation Accuracy: 80.60%, Macro F1-Score: 0.8021
Epoch 19/21, Loss: 0.1039
Validation Accuracy: 82.60%, Macro F1-Score: 0.8237
Epoch 20/21, Loss: 0.1604
Validation Accuracy: 80.20%, Macro F1-Score: 0.7973
Epoch 21/21, Loss: 0.1296
Validation Accuracy: 83.40%, Macro F1-Score: 0.8221
Training complete.
```

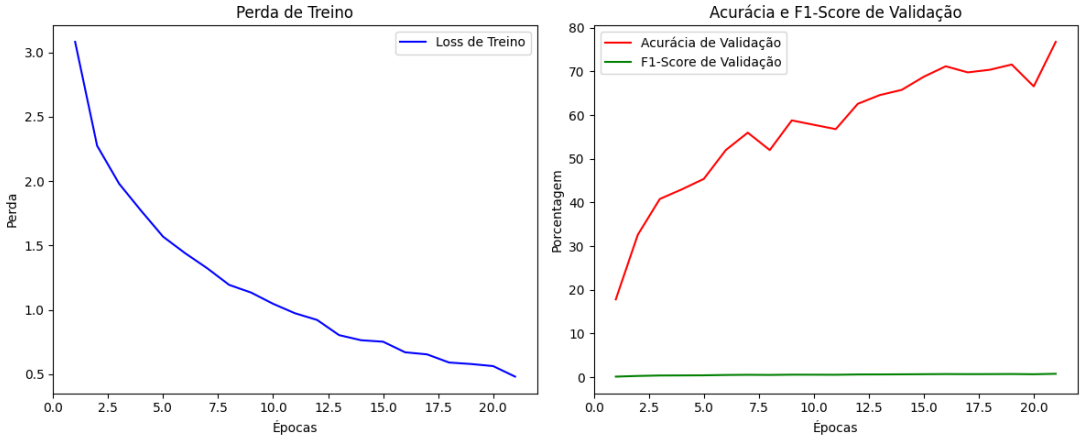

Figura 1 – Primeira fase de treino

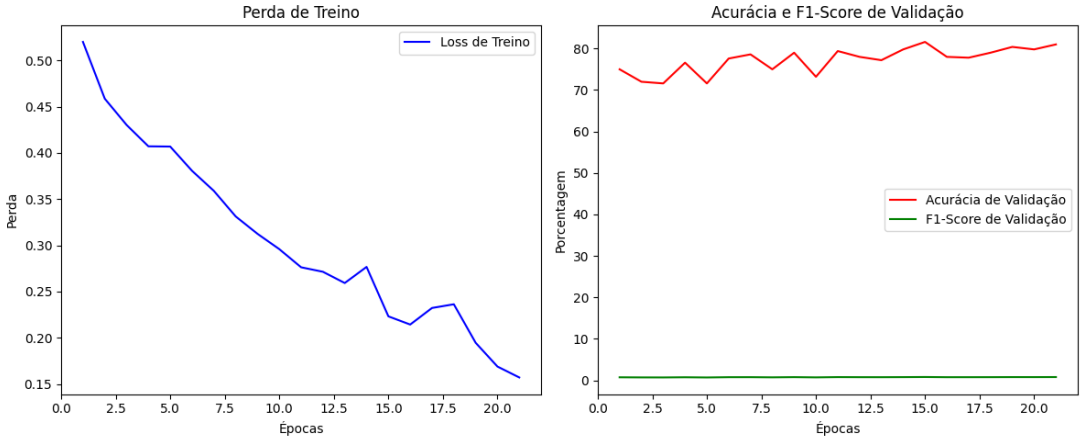

Figura 2 – Segunda Fase de Treino

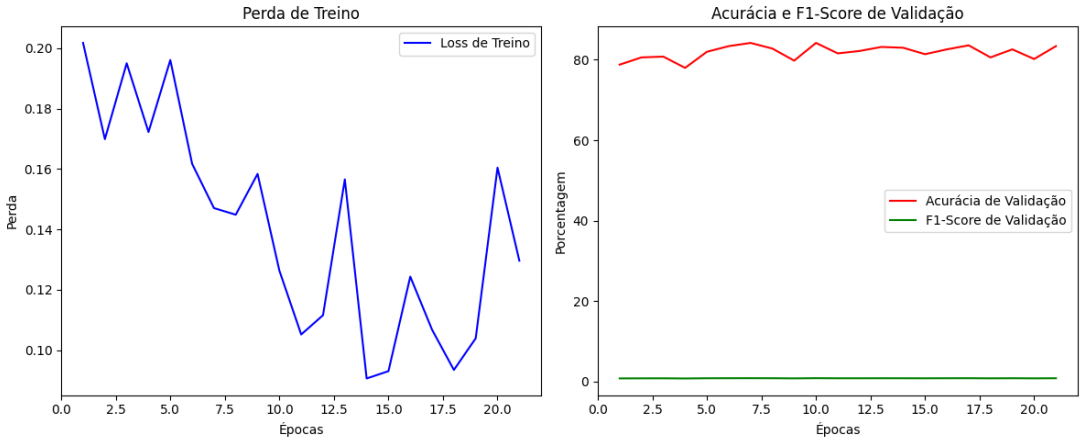

Figura 3 – Terceira Fase de Treino

8 DESCRIÇÃO DO APRENDIZADO COM O TRABALHO

Foi realizado treino de 2 modelos, o primeiro com Dropout(0.5) e o segundo que apresentou melhor resultado sem o uso de Dropout.

Outro ponto que foi observado com uma grande melhora na na avaliação foi o uso de dos data transform informados, onde tivemos uma grande melhora a avaliação do modelo.

Descrição das dificuldades: Grande parte do desafio foi a separação dos dados de treino e a filtragem das classes para o treino, ea adaptação dos dados e camadas de saiada do modelo para as 50 classes selecionadas.

9 REFERENCIAS

https://github.com/FabricioMT/Trabalho-CNN
https://pytorch.org/vision/0.18/generated/torchvision.datasets.Flowers102.html
https://pytorch.org
https://scikit-learn.org