CNN: Fine-Tunning com WNN e HDC

Grupo: Caio Guimarães, Eduardo Loivos

Problema

Nós escolhemos resolver o problema de classificação de cores de veículos a partir de imagens utilizando ***CNN***.

O dataset escolhido para o problema foi o ***Vehicle-Rear*** da UFPR, que foi utilizado no artigo "Vehicle-Rear: A New Dataset to Explore Feature Fusion for Vehicle Identification Using Convolutional Neural Networks".

Esse dataset é um conjunto de dados que contém mais de três horas de vídeos em alta resolução, com informações precisas sobre marca, modelo, cor e ano de quase 3.000 veículos, além da posição e identificação de suas placas.

As sequências temporais mostram exemplos de (a) motocicletas; (b) carros e ônibus; (c) caminhões; (a) e (c) em condições climáticas normais; (b) quadros escuros causados pelo movimento de veículos de grande porte; e (d) condições severas de iluminação

As arquiteturas, os modelos treinados e o conjunto de dados estão disponíveis no repositório https://github.com/icarofua/vehicle-rear

Implementação

Adaptação para PyTorch

O primeiro passo a ser feito foi a adaptação do modelo para PyTorch. Alteramos a etapa de pré-processamento para utilizar os ***transforms*** e a forma com que o dataset é carregado foi alterada para a utilização do ***DataLoader***.

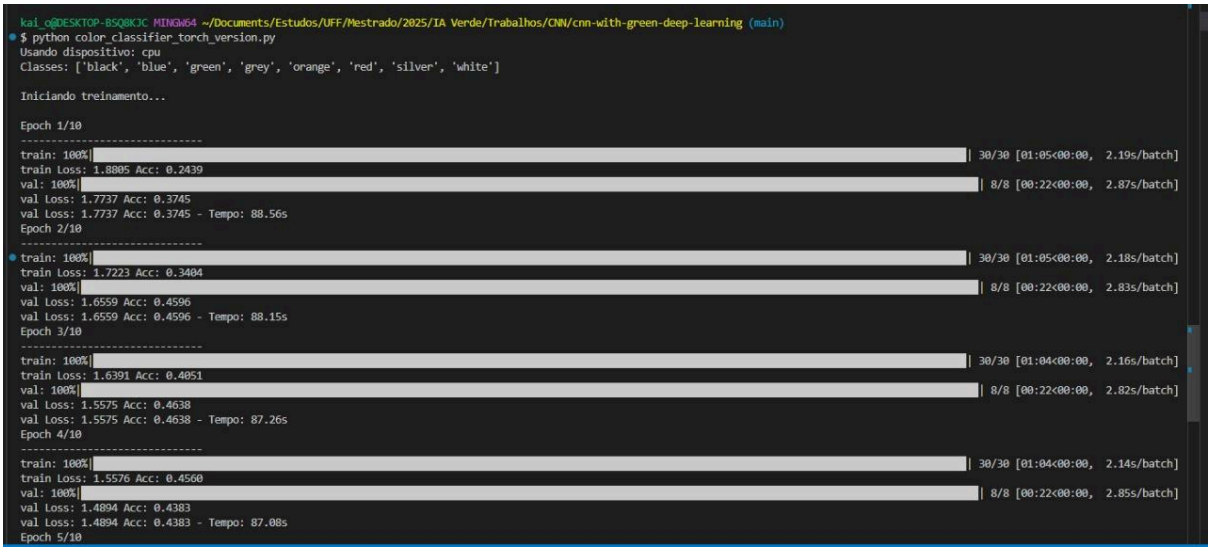
```python
# Transforms
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(img_size[0]),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],      You, 3 days ago • Cria uma versão do m
                             [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(img_size[0]),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
}

# Carrega datasets
image_datasets = {
    x: datasets.ImageFolder(os.path.join(data_dir),
                            transform=data_transforms[x])
    for x in ['train', 'val']
}

dataset_size = len(image_datasets['train'])
val_size = int(0.2 * dataset_size)
train_size = dataset_size - val_size
train_dataset, val_dataset = random_split(image_datasets['train'], [train_size, val_size])

dataloaders = {
    'train': DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4),
    'val': DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=4)
}
```

Treinamento

A escolha de utilizar a rede neural ***ResNet50*** pré-treinada com pesos do ImageNet foi mantida e o treinamento foi feito no arquivo ***color_classifier_torch_version.py***.

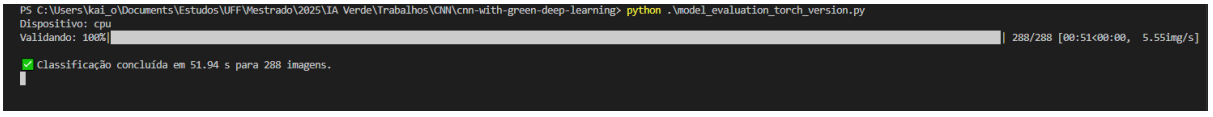
```python
# Carrega a ResNet50 pré-treinada
model = models.resnet50(weights='IMAGENET1K_V2')
for param in model.parameters():
    param.requires_grad = False

# Substitui a última camada totalmente conectada
model.fc = nn.Sequential(
    nn.Linear(model.fc.in_features, 128),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(128, num_classes)
)

model = model.to(device)

# Otimizador e loss
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=1e-4)
```

O ***PyTorch*** não tem um método ***.fit()*** igual ao ***Keras***, então foi necessário criar o loop de treino por épocas de forma manual.

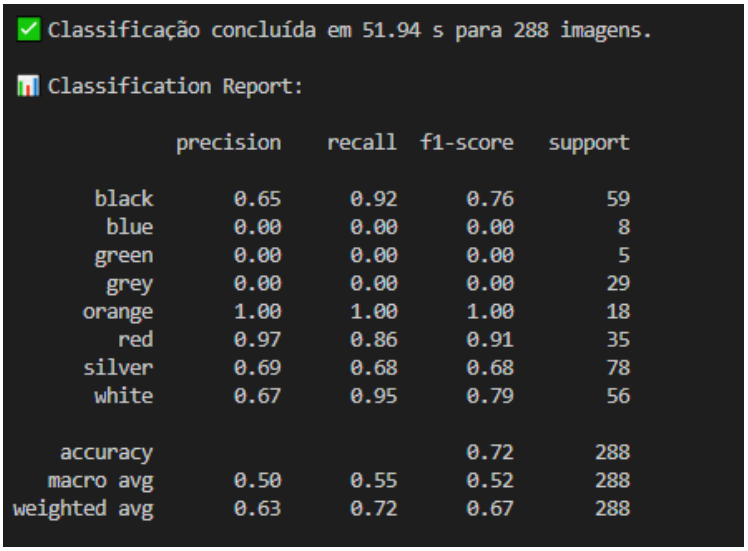


Resultados

A avaliação do modelo também precisou ser adaptada para utilizar o ***PyTorch***, ela foi feita no arquivo ***model_evaluation_torch_version.py***.

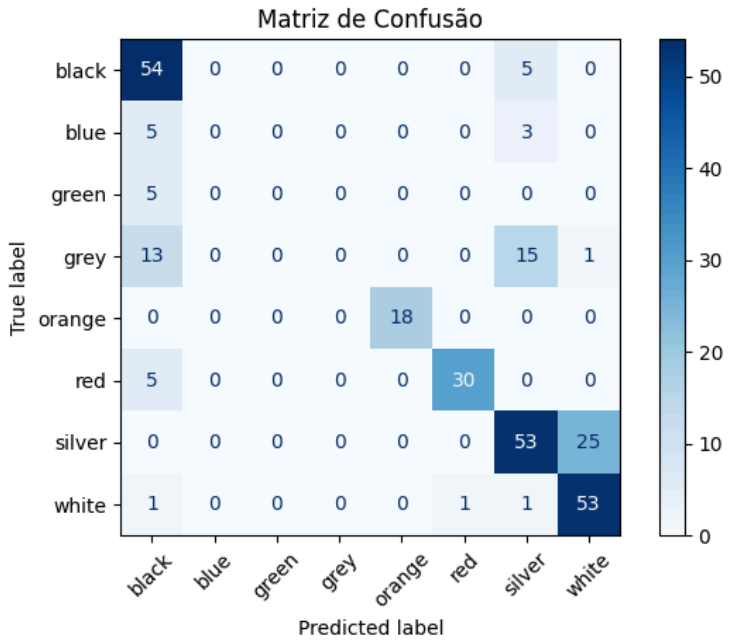


A partir da predição, podemos gerar o relatório de métricas e realizar sua análise.



Podemos perceber que o modelo apresenta uma acurácia de 72%, levemente inferior ao modelo com ***TensorFlow***, no entanto,a dificuldade em prever as cores azul e verde se manteve.

Por último, é gerada uma matriz de confusão para análise dos resultados.



Técnicas de Fine-Tuning

Realização de fine-tuning utilizando classificadores baseados na WiSARD e HDC.

Para realizar esse procedimento o primeiro passo foi recuperar o resultado da última camada de convolução, que contém o vetor de features e utilizá-la como a entrada dos novos classificadores

WiSARD

Iniciamos a implementação carregando o modelo base pré-treinado ***ResNet50***, e então aplicamos o mesmo pipeline do treinamento e então carregamos os pesos treinados. Para utilizar ter como resultado somente o vetor de features da ***CNN***, nós removemos a última camada. Nós então utilizamos o ***transform*** para realizar o pré-processamento das imagens.

```python
# Modelo
model = models.resnet50(weights='IMAGENET1K_V2')
for param in model.parameters():
    param.requires_grad = False

model.fc = nn.Sequential(
    nn.Linear(model.fc.in_features, 128),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(128, num_classes)
)
model.load_state_dict(torch.load(model_path, map_location=device))      You, 12 hours ago
model.to(device)
model.eval()

# Extrator de features (sem o otimizador e loss)
feature_extractor = nn.Sequential(*list(model.children())[:-1])
feature_extractor.to(device)
feature_extractor.eval()

# Dataset
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

train_dataset = datasets.ImageFolder("data", transform=transform)
test_dataset  = datasets.ImageFolder("data_test", transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=False, num_workers=0)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False, num_workers=0)
```

Para realizar a classificação com o ***WiSARD*** foi necessário unir os batches em um único tensor e realizar a codificação com Termômetro. Além disso, realizamos a redução da dimensão através da técnica de ***PCA*** (Principal component analysis)

```python
#  Extrair features
def extract_features(dataloader):
    all_features = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs = inputs.to(device)
            feats = feature_extractor(inputs)
            feats = torch.flatten(feats, 1)

            all_features.append(feats.cpu())
            all_labels.extend(labels.cpu())

    # Junta todos os batches em um único tensor
    X = torch.cat(all_features, dim=0)
    y = torch.tensor(all_labels)

    return X, y
```

```python
# Reduz a dimensão para 32
pca = PCA(n_components=32)
X_train_pca = pca.fit_transform(X_train)
X_test_pca  = pca.transform(X_test)

# Codificação com Termômetro
bits_encoding = 20
encoding = Thermometer(bits_encoding).fit(X_train_pca)
X_train_bin = encoding.binarize(X_train_pca).flatten(start_dim=1)
encoding = Thermometer(bits_encoding).fit(X_test_pca)
X_test_bin = encoding.binarize(X_test_pca).flatten(start_dim=1)
```

```python
# WiSARD
entry_size = X_train_bin.shape[1]
tuple_size = 16

with torch.no_grad():
    wisard = Wisard(entry_size, num_classes, tuple_size, bleaching=True)
    wisard.fit(X_train_bin, y_train)

# Avaliação
y_pred = wisard.predict(X_test_bin)
```

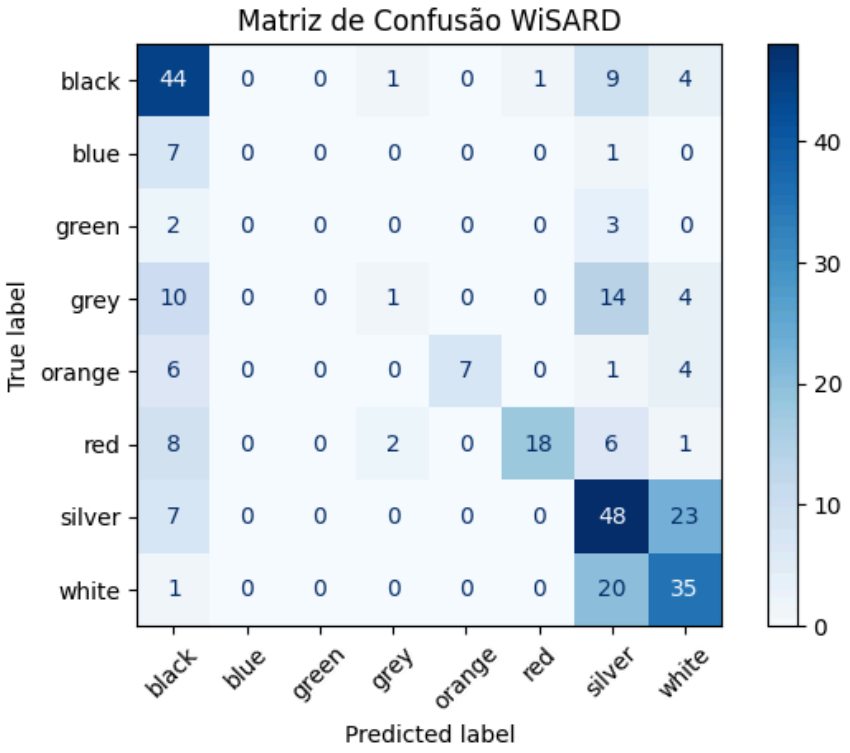
A execução do treinamento e avaliação do modelo foi realizada no arquivo ***color_classifier_WiSARD.py***

```
kai_g@DESKTOP-BSQ8KJC MINGW64 ~/Documents/Estudos/UFF/Mestrado/2025/IA Verde/Trabalhos/CNN/cnn-with-green-deep-learning (main)
$ python color_classifier_WiSARD.py
Dispositivo: cpu
🔍 Extraindo features de treino...
✅ Treino: torch.Size([1178, 160])
🔍 Extraindo features de teste...
✅ Teste: torch.Size([288, 160])

📊 Classification Report:
```

A partir da predição, podemos gerar o relatório de métricas e realizar sua análise.

	precision	recall	f1-score	support
black	0.52	0.75	0.61	59
blue	0.00	0.00	0.00	8
green	0.00	0.00	0.00	5
grey	0.25	0.03	0.06	29
orange	1.00	0.39	0.56	18
red	0.95	0.51	0.67	35
silver	0.47	0.62	0.53	78
white	0.49	0.62	0.55	56
accuracy			0.53	288
macro avg	0.46	0.37	0.37	288
weighted avg	0.53	0.53	0.50	288

Podemos perceber que o modelo apresenta uma acurácia de 53%. Por último, é gerada uma matriz de confusão para análise dos resultados.



HDC

Nós escolhemos a biblioteca ***binhd*** para realizar o fine-tuning com HDC, a primeira etapa é recuperar os pesos da camada de convolução do modelo treinado anteriormente. Para isso é necessário recriar a estrutura do modelo segundo o código:

```
# Carregar ResNet50 para extração de features
model = models.resnet50(weights='IMAGENET1K_V2')
for param in model.parameters():
    param.requires_grad = False

model.fc = nn.Sequential(
    nn.Linear(model.fc.in_features, 128),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(128, num_classes)
)
```

E em seguida carregar os pesos do treinamento com a camada final MLP.

```
model.load_state_dict(torch.load(model_path, map_location=device))
model.to(device)
model.eval()
```

Por fim, devemos descartar a camada sequencial para substituí-la por uma camada HDC.

```
# Extrator de features (sem a camada fc)
feature_extractor = nn.Sequential(*list(model.children())[:-1])
feature_extractor.to(device)
feature_extractor.eval()
```

Outra parte fundamental para o uso do HDC é a codificação com o ***Record-Based Encoder***.

```
# Codificacao HDC com record-based encoding
class RecordEncoder(nn.Module):
    def __init__(self, out_features, size, levels, low, high):
        super(RecordEncoder, self).__init__()
        self.position = embeddings.Random(size, out_features, vsa="BSC", dtype=torch.uint8)
        self.value = ScatterCode(levels, out_features, low=low, high=high)

    def forward(self, x):
        sample_hv = torchhd.bind(self.position.weight, self.value(x))
        sample_hv = torchhd.multiset(sample_hv)
        return sample_hv

record_encoder = RecordEncoder(hd_dim, 128, num_levels, min_val, max_val).to(device)
```

Definição dos hiperparâmetros:
Dimensões do Hipervetor = 1000
Quantidade de níveis do ScatterCode = 100

Devido a alta dimensionalidade das entradas foi necessário dividi-las em batches e aplicar uma camada de ***PCA*** (Principal component analysis) para selecionar as features mais relevantes.

```
# Aplicar PCA
print("📉 Reduzindo dimensionalidade com PCA...")
pca = PCA(n_components=128)
```

O código a seguir realiza o pré Processamento em batches e treina a camada de classificação HDC.

```
# Treinamento em batches
print("\n🚀 Treinando em batches...")
record_encoder.eval()
feature_extractor.eval()

X_train_batches = []
y_train_batches = []

with torch.no_grad():
    for batch_idx, (inputs, labels) in enumerate(train_loader):
        print(f"📦 Processando batch {batch_idx + 1}...")
        inputs = inputs.to(device)
        feats = feature_extractor(inputs)
        feats = torch.flatten(feats, 1).cpu()
        feats = torch.from_numpy(pca.transform(feats))
        encoded_feats = record_encoder(feats.to(device))

        X_train_batches.append(encoded_feats)
        y_train_batches.append(labels)

X_train_tensor = torch.tensor(np.concatenate(X_train_batches), dtype=torch.int8)
y_train_tensor = torch.cat(y_train_batches, dim=0)

print("🧠 Treinando classificador HDC...")
hdc_model.fit(X_train_tensor, y_train_tensor)
```

Em seguida, o mesmo processo é realizado com base de dados de teste.

```
# Predição em batches
y_true, y_pred = [], []
print("\n🔍 Predizendo em batches...")
with torch.no_grad():
    for batch_idx, (inputs, labels) in enumerate(test_loader):
        print(f"🔄 Predizendo batch {batch_idx + 1}...")
        inputs = inputs.to(device)
        feats = feature_extractor(inputs)
        feats = torch.flatten(feats, 1).cpu()
        feats = torch.from_numpy(pca.transform(feats))
        encoded_feats = record_encoder(feats.to(device))

        preds = hdc_model.predict(encoded_feats)
        y_true.extend(labels.numpy())
        y_pred.extend(preds)

# Avaliação
print("\n📊 Classification Report:\n")
print(classification_report(y_true, y_pred, target_names=class_names))
```

Os resultados do experimento foram:

	precision	recall	f1-score	support
black	0.20	0.32	0.25	59
blue	0.00	0.00	0.00	8
green	0.00	0.00	0.00	5
grey	0.00	0.00	0.00	29
orange	0.00	0.00	0.00	18
red	0.00	0.00	0.00	35
silver	0.31	0.40	0.35	78
white	0.22	0.34	0.26	56
accuracy			0.24	288
macro avg	0.09	0.13	0.11	288
eighted avg	0.17	0.24	0.20	288

Matriz de confusão do experimento: