Convolution Neural Networks

Equipo:

- Integrante 1 (XX%)
- Integrante 2 (XX%)
- Integrante 3 (XX%)
- Integrante 4 (XX%)

Objetivo:

Comprender e implementar una Red Neuronal Convolucional (CNN) para clasificación de imágenes. El objetivo es construir un modelo capaz de distinguir entre categorías con cierta complejidad, lo cual requiere un uso cuidadoso de los principios de las CNN.

Tareas:

1. **Exploración y preprocesamiento de datos**: Normalizar los datos (restar la media, dividir por la desviación estándar) y aplicar aumento de datos (volteos, rotaciones, etc.).
2. **Diseño de la arquitectura CNN**: Diseñar una CNN con al menos una capa convolucional, seguidas de max pooling, y una o más capas totalmente conectadas. Establecer hiperparámetros adecuados como tamaño del kernel, stride y número de filtros. Experimentar con diferentes funciones de activación y tasas de aprendizaje, documentando tus elecciones y el razonamiento detrás de ellas. Recuerda que nuestra misión es reducir el número de parámetros. (máx. 100000)
3. **Entrenamiento y evaluación**: Entrenar la CNN y monitorear la pérdida y precisión tanto en el conjunto de entrenamiento como en el de validación. Asegurarse de que el tamaño de lote, número de épocas y tasa de aprendizaje sean ajustables.
4. **Reflexión y análisis**: Analizar los resultados, discutiendo posibles mejoras y los desafíos encontrados durante el entrenamiento.

Entregables

1. **Canvas**: Entregar tu Notebook de Jupyter/Colab, incluyendo la arquitectura final del modelo, registros del entrenamiento, matriz de confusión y un breve análisis.
2. **Foro**: Publicar la matriz de confusión como imagen y un resumen de la arquitectura final de tu modelo junto con su precisión final.

Parte 1: Configuración e Importaciones

```
# Import necessary libraries
import os
import shutil
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Set device for computation (GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

...

Parte 2: Carga y Preprocesamiento de Datos

Comienza a programar o generar con IA.

```
# Download and unzip Tiny ImageNet if not included directly
!wget -q http://cs231n.stanford.edu/tiny-imagenet-200.zip
!unzip -qq tiny-imagenet-200.zip


val_dir = './tiny-imagenet-200/val'
images_dir = os.path.join(val_dir, 'images')
ann_file = os.path.join(val_dir, 'val_annotations.txt')

with open(ann_file) as f:
    for line in f:
        name, wnid = line.split('\t')[:2]
        os.makedirs(os.path.join(val_dir, wnid), exist_ok=True)
        shutil.move(os.path.join(images_dir, name), os.path.join(val_dir, wnid,

shutil.rmtree(images_dir)

TRAIN_DIR = './tiny-imagenet-200/train/'
VALID_DIR = './tiny-imagenet-200/val/'

IMG_SIZE = 64
```

```
BATCH_SIZE = 32

train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

val_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
])

train_dataset = datasets.ImageFolder(root=TRAIN_DIR, transform=train_transform)
val_dataset = datasets.ImageFolder(root=VALID_DIR, transform=val_transform)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, r
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False, num_

print(f"Entrenamiento: {len(train_dataset)} imágenes, {len(train_dataset.classe
print(f"Validación: {len(val_dataset)} imágenes, {len(val_dataset.classes)} cla
```

```
...
```

```
for inputs, labels in train_loader:
    print(labels)
    break
```

```
...
```

```
len(train_loader)
```

```
...
```

```
from sklearn.model_selection import StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n_splits=10, test_size=None, random_state=0)
cmp_indices, subset_indices = next(sss.split(train_dataset.imgs, [label for _,
```

```
(cmp_indices.shape, subset_indices.shape)
```

```
...
```

```
subset_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, sampler=torch.
print(len(subset_loader))
```

```
...
```

Parte 3: Definir el Modelo CNN

```
# Define a simple CNN architecture
# pytorch / tensorflow
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 200)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 128 * 8 * 8)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = CNNModel().to(device)
```

Paso 4: Definir la Función de Pérdida y el Optimizador

```
# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Paso 5: Entrenamiento del Modelo

```
# Training loop
EPOCHS = 10  # Adjust based on available time

train_loss, val_loss = [], []

for epoch in range(EPOCHS):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        break
    train_loss.append(running_loss / len(train_loader))

    # Validation loss
    model.eval()
    running_val_loss = 0.0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_val_loss += loss.item()
    val_loss.append(running_val_loss / len(val_loader))

    print(f'Epoch [{epoch+1}/{EPOCHS}], Train Loss: {train_loss[-1]:.4f}, Val L
```

...

Paso 6: Graficar la Pérdida de Entrenamiento y Validación

```
# Plotting training and validation loss
plt.plot(train_loss, label='Train Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```