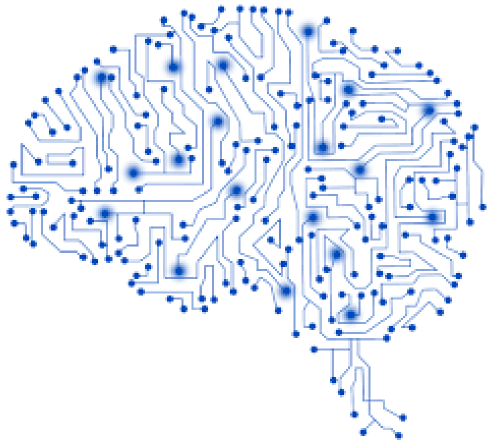
University of Minho
School of Engineering



Aprendizagem Profunda

Deep Neural Network – CNN Multiclass

APP @ MEI/4º ano – 2º Semestre
Victor Alves, Miguel Rocha

Part III

Hands On

CNN for multiclass image classification

MNIST (Modified National Institute of Standards and Technology) Dataset

It is considered the "hello world" dataset of *computer vision*.

- Dataset of manually written digit images
- Inputs: 28x28 pixels images
- Output: class representing the digit (10 classes, digits 0-9)
- 70k images of which 60k are for training and 10k for testing
- 2 attributes: the image id and its label

We will use a convolutional neural network to classify the digit in each 28x28 image.

0. Prepare the setup

Install pytorch (if needed)

Imports

Constants

Device management (optional)

0. Prepare the setup

```
device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")


def get_default_device():
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)
```

```
class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

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

device = get_default_device()
print(device)
```

1. Prepare the data

```
train_transform = Compose(
        [ToTensor(),
         Normalize(mean=(0.1307,), std=(0.3081,))
        ])
test_transform = Compose(
        [ToTensor(),
         Normalize(mean=(0.1307,), std=(0.3081,))
        ])
```

1. Prepare the data

```
class CSVDataset(Dataset):

    def __init__(self, path, transform=None):
        self.transform = transform
        df_set = pd.read_csv(path, header=0)
        self.x = df_set.values[:, 1:]
        self.y = df_set.values[:, 0]
        self.x = self.x.astype('float32')
        self.y = self.y.astype('long')

    def __len__(self):
        return len(self.x)
```

```
def __getitem__(self, idx):
    label = self.y[idx]
    image = self.x[idx]
    if self.transform is not None:
        image = self.transform(image)
    return image, label

def get_TensorDataset(self):
    x = self.x.reshape(len(self.x), 1, 28, 28)
    xmax, xmin = x.max(), x.min()
    x = (x - xmin)/(xmax - xmin)
    x  = torch.from_numpy(np.array(x)).float()
    y  = torch.from_numpy(np.array(self.y)).type(
            torch.LongTensor)
    cases = torch.utils.data.TensorDataset(x,y)
    return cases
```

1. Prepare the data

```
def prepare_data_loaders(path_train, path_test):
    dataset_train = CSVDataset(path_train,transform=train_transform)
    dataset_test = CSVDataset(path_test,transform=test_transform)
    train = dataset_train.get_TensorDataset()
    train_size = int(0.8 * len(train))
    val_size = len(train) - train_size
    train, validation = random_split(train, [train_size, val_size], generator=torch.Generator().manual_seed(42))
    test = dataset_test.get_TensorDataset()
    train_dl = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
    val_dl = DataLoader(validation, batch_size=BATCH_SIZE, shuffle=True)
    test_dl = DataLoader(test, batch_size=BATCH_SIZE, shuffle=True)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=False)
    val_dl_all = DataLoader(validation, batch_size=len(validation), shuffle=True)
    test_dl_all = DataLoader(test, batch_size=len(test), shuffle=False)
    return train_dl, val_dl, test_dl, train_dl_all, val_dl_all, test_dl_all

train_dl, val_dl, test_dl, train_dl_all, val_dl_all, test_dl_all = prepare_data_loaders(PATH_TRAIN, PATH_TEST)
```

1.1 Visualize the data

```
def output_label(label,mapping='label'):
    if mapping == 'ext':
        output_mapping = { 0:"zero", 1:"um", 2:"dois", 3:"tres", 4:"quatro", 5:"cinco", 6:"seis", 
7:"sete",
                           8:"oito", 9:"nove" }
    elif mapping == 'ext2':
        output_mapping = { "0":"zero", "1":"um", "2":"dois", "3":"tres", "4":"quatro", "5":"cinco",
                           "6":"seis", "7":"sete", "8":"oito", "9":"nove" }
    else:
        output_mapping = { 0: "0", 1: "1", 2: "2", 3: "3", 4: "4", 5: "5", 6: "6", 7: "7", 8: "8", 9: "9"}
    input = (label.item() if type(label) == torch.Tensor else label)
    return output_mapping[input]
```

1.1 Visualize the data

```
from IPython.display import display
def visualize_data(path):
    df = pd.read_csv(path, header=0)
    display(df)
```

```
def visualize_dataset(train_dl, test_dl):
    print(f"Quantidade de casos de Treino:{len(train_dl.dataset)}")
    print(f"Quantidade de casos de Validação:{len(val_dl.dataset)}")
    print(f"Quantidade de casos de Teste:{len(test_dl.dataset)}")
    x, y = next(iter(train_dl))
    print(f"Shape tensor batch casos treino, input: {x.shape}, output: {y.shape}")
    x, y = next(iter(val_dl))
    print(f"Shape tensor batch casos validação, input: {x.shape}, output: 
{y.shape}")
    x, y = next(iter(test_dl))
    print(f"Shape tensor batch casos test, input: {x.shape}, output: {y.shape}")
    print(f'Valor máximo:{torch.max(x)} Valor mínimo:{torch.min(x)}')
    x=x.detach().numpy()
    print(f'Valor máximo:{np.max(x)} Valor mínimo:{np.min(x)}')
    print(y)

visualize_data(PATH_TRAIN)
```

1.1 Visualize the data

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59995	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59996	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59997	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59999	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

60000 rows × 785 columns

```
Quantidade de casos de Treino:48000
Quantidade de casos de Validação:12000
Quantidade de casos de Teste:10000
Shape tensor batch casos treino, input: torch.Size([32, 1, 28, 28]), output: torch.Size([32])
Shape tensor batch casos validação, input: torch.Size([32, 1, 28, 28]), output: torch.Size([32])
Shape tensor batch casos test, input: torch.Size([32, 1, 28, 28]), output: torch.Size([32])
Valor máximo:1.0 Valor mínimo:0.0
Valor máximo:1.0 Valor mínimo:0.0
tensor([7, 1, 7, 4, 3, 4, 2, 8, 7, 4, 2, 5, 8, 3, 2, 5, 1, 1, 5, 4, 3, 2, 1, 2,
        3, 9, 5, 6, 7, 7, 3, 9])
```

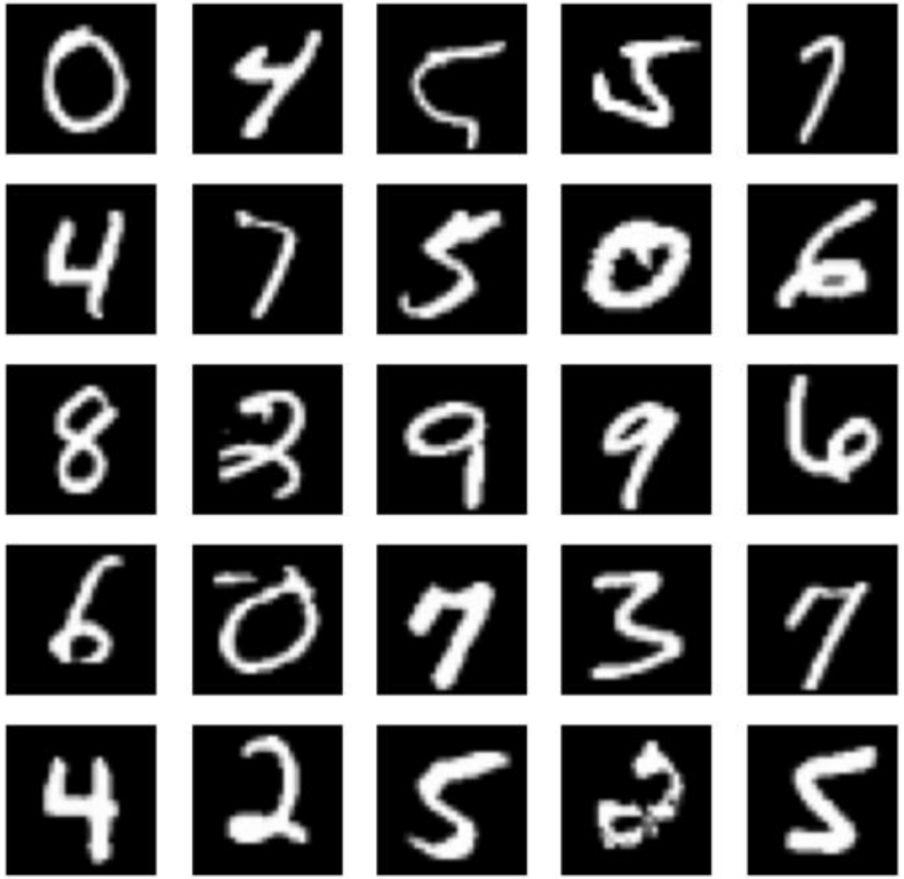
1.1 Visualize the data

```
def visualize_mnist_images(dl):
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    plt.figure(figsize=(8,8))
    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        plt.imshow(inputs[i][0], cmap='gray')
    plt.show()

visualize_mnist_images(train_dl)
```

1.1 Visualize the data

```
def visualize_mnist_images(dl):
    i, (inputs, targets) = next(enumerate(dl))
    print(inputs.shape)
    plt.figure(figsize=(8,8))
    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.axis('off')
        plt.grid(b=None)
        plt.imshow(inputs[i][0], cmap='gray')
    plt.show()

visualize_mnist_images(train_dl)
```

```
torch.Size([32, 1, 28, 28])
```



1.2 Verify dataset balancing

2. Define the model (1)

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

```
class CNNModel_1(Module):
    def __init__(self):
        super(CNNModel_1, self).__init__()
        self.layer1 = Sequential(Conv2d(in_channels=1, out_channels=32, kernel_size=(3,3)),
                                 ReLU(),
                                 MaxPool2d(kernel_size=(2,2), stride=(2,2))
                                )
        self.layer2 = Sequential(Conv2d(in_channels=32, out_channels=32, kernel_size=(3,3)),
                                 ReLU(),
                                 MaxPool2d(kernel_size=(2,2), stride=(2,2))
                                )
        self.fc1 = Linear(in_features=5*5*32, out_features=100)
        kaiming_uniform_(self.fc1.weight, nonlinearity='relu')
        self.act1 = ReLU()
        self.fc2 = Linear(in_features=100, out_features=10)
        xavier_uniform_(self.fc2.weight)
        self.act2 = Softmax(dim=1)
```

2. Define the model (1)

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(-1, 4*4*50)
        out = self.fc1(out)
        out = self.act1(out)
        out = self.fc2(out)
        out = self.act2(out)
        return out

model = CNNModel_1()
print(summary(model, input_size=(BATCH_SIZE, 1,28,28), verbose=0))
model.to(device)
```

2. Define the model (1)

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [32, 32, 13, 13]          --
│    └─Conv2d: 2-1                       [32, 32, 26, 26]          320
│    └─ReLU: 2-2                         [32, 32, 26, 26]          --
│    └─MaxPool2d: 2-3                    [32, 32, 13, 13]          --
├─Sequential: 1-2                        [32, 32, 5, 5]            --
│    └─Conv2d: 2-4                       [32, 32, 11, 11]          9,248
│    └─ReLU: 2-5                         [32, 32, 11, 11]          --
│    └─MaxPool2d: 2-6                    [32, 32, 5, 5]            --
├─Linear: 1-3                            [32, 100]                 80,100
├─ReLU: 1-4                              [32, 100]                 --
├─Linear: 1-5                            [32, 10]                  1,010
├─Softmax: 1-6                           [32, 10]                  --
==========================================================================================
Total params: 90,678
Trainable params: 90,678
Non-trainable params: 0
Total mult-adds (M): 54.24
==========================================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 6.56
Params size (MB): 0.36
Estimated Total Size (MB): 7.02
==========================================================================================
```

3. Train the model


```
def train_model(h5_file, train_dl, val_dl, model, criterion, optimizer):
    liveloss = PlotLosses()
    for epoch in range(EPOCHS):
        logs = {}
        model.train()
        running_loss  = 0.0
        running_corrects  = 0.0
        for inputs, labels in train_dl:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            running_loss += loss.detach() * inputs.size(0)
            _, preds = torch.max(outputs, 1)
            running_corrects += torch.sum(preds == labels.data)
        (...)
```

```
        (...)
        epoch_loss = running_loss / len(train_dl.dataset)
        epoch_acc = running_corrects.float()/len(train_dl.dataset)
        logs['loss'] = epoch_loss.item()
        logs['accuracy'] = epoch_acc.item()

        model.eval()
        running_loss  = 0.0
        running_corrects  = 0.0
        for inputs, labels in val_dl:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            running_loss += loss.detach() * inputs.size(0)
            _, preds = torch.max(outputs, 1)
            running_corrects += torch.sum(preds == labels.data)
        epoch_loss = running_loss / len(val_dl.dataset)
        epoch_acc = running_corrects.float() / len(val_dl.dataset)
        logs['val_loss'] = epoch_loss.item()
        logs['val_accuracy'] = epoch_acc.item()
        liveloss.update(logs)
        liveloss.send()
    torch.save(model,h5_file)
```

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

3. Train the model (1)

For model 1:

```
model = CNNModel_1()
print(summary(model, input_size=(BATCH_SIZE, 1,28,28), verbose=0))
model.to(device)

EPOCHS = 15
LEARNING_RATE = 0.001
criterion = CrossEntropyLoss()
optimizer = SGD(model.parameters(), lr=LEARNING_RATE)
starttime = time.perf_counter()
train_model('CNNModel_1.pth', train_dl, val_dl, model, criterion, optimizer)
endtime = time.perf_counter()
print(f"Tempo gasto: {endtime - starttime} segundos")
```

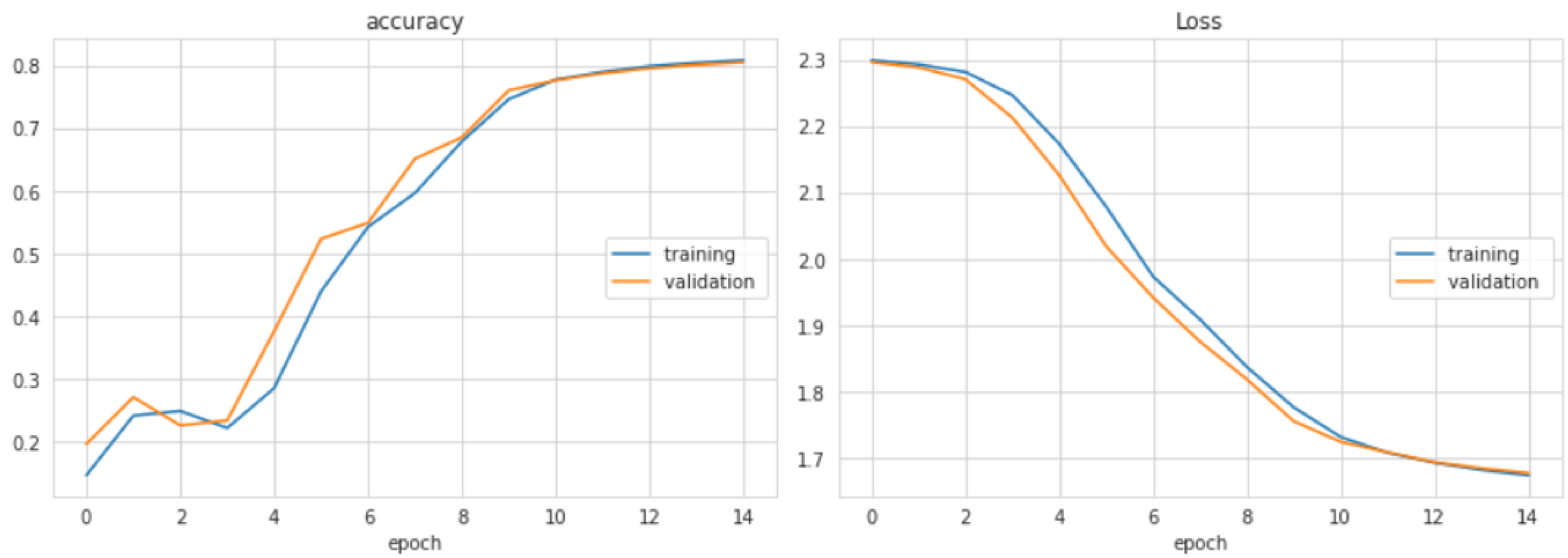
Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

3. Train the model (1)


```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [32, 32, 13, 13]          --
│    └─Conv2d: 2-1                       [32, 32, 26, 26]          320
│    └─ReLU: 2-2                         [32, 32, 26, 26]          --
│    └─MaxPool2d: 2-3                    [32, 32, 13, 13]          --
├─Sequential: 1-2                        [32, 32, 5, 5]            --
│    └─Conv2d: 2-4                       [32, 32, 11, 11]          9,248
│    └─ReLU: 2-5                         [32, 32, 11, 11]          --
│    └─MaxPool2d: 2-6                    [32, 32, 5, 5]            --
├─Linear: 1-3                            [32, 100]                 80,100
├─ReLU: 1-4                              [32, 100]                 --
├─Linear: 1-5                            [32, 10]                  1,010
├─Softmax: 1-6                           [32, 10]                  --
==========================================================================================
Total params: 90,678
Trainable params: 90,678
Non-trainable params: 0
Total mult-adds (M): 54.24
==========================================================================================
Input size (MB): 0.10
Forward/backward pass size (MB): 6.56
Params size (MB): 0.36
Estimated Total Size (MB): 7.02
==========================================================================================
```

3. Train the model (1)

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax




```
accuracy
        training                 (min:    0.147, max:    0.808, cur:    0.808)
        validation               (min:    0.197, max:    0.804, cur:    0.804)
Loss
        training                 (min:    1.674, max:    2.299, cur:    1.674)
        validation               (min:    1.678, max:    2.296, cur:    1.678)
Tempo gasto: 2236.4488256331533 segundos
```

4. Evaluate the model


```
def evaluate_model(test_dl, model):
    predictions = list()
    actual_values = list()
    for inputs, labels in test_dl:
        inputs = inputs.to(device)
        labels = labels.to(device)
        yprev = model(inputs)
        yprev = yprev.detach().cpu().numpy()
        actual = labels.cpu().numpy()
        yprev = np.argmax(yprev, axis=1)
        actual = actual.reshape((len(actual), 1))
        yprev = yprev.reshape((len(yprev), 1))
        predictions.append(yprev)
        actual_values.append(actual)
        break
    predictions, actual_values = np.vstack(predictions), np.vstack(actual_values)
    return actual_values, predictions
```

4. Evaluate the model


```
def display_predictions(actual_values, predictions):
    acertou=0
    falhou = 0
    primeiros=0
    for r,p in zip(actual_values, predictions):
        if primeiros <20:
            print(f'real:{r} previsão:{p}')
            primeiros +=1
        if r==p: acertou+=1
        else: falhou+=1
    corrects = np.sum(predictions == actual_values)
    acc = corrects/len(test_dl.dataset)
    (...)
```

```
    (...)
    acc = accuracy_score(actual_values, 
predictions)
    print(f'Accuracy: {acc:0.3f}\n')
    print(f'acertou:{acertou} falhou:{falhou}')
    acc = accuracy_score(actual_values, 
predictions)
    print(f'Accuracy: {acc:0.3f}\n')
    print(f'acertou:{acertou} falhou:{falhou}')
```

4. Evaluate the model


```
def display_confusion_matrix(cm,list_classes):
    plt.figure(figsize = (16,8))
    sns.heatmap(cm,annot=True,xticklabels=list_classes,yticklabels=list_classes, annot_kws={"size": 12},
                      fmt='g', linewidths=.5)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

model= torch.load('CNNModel_1.pth')
actual_values, predictions = evaluate_model(test_dl_all, model)
display_predictions(actual_values, predictions )
print(classification_report(actual_values, predictions))
cr =classification_report(actual_values, predictions, output_dict=True)
list_classes=[output_label(n,'ext2') for n in list(cr.keys())[0:10] ]
cm = confusion_matrix(actual_values, predictions)
print (cm)
display_confusion_matrix(cm,list_classes)
```

4. Evaluate the model


```
real:[7] previsão:[7]
real:[2] previsão:[2]
real:[1] previsão:[1]
real:[0] previsão:[0]
real:[4] previsão:[4]
real:[1] previsão:[1]
real:[4] previsão:[4]
real:[9] previsão:[9]
real:[5] previsão:[5]
real:[9] previsão:[9]
real:[0] previsão:[0]
real:[6] previsão:[6]
real:[9] previsão:[9]
real:[0] previsão:[0]
real:[1] previsão:[1]
real:[5] previsão:[5]
real:[9] previsão:[9]
real:[7] previsão:[7]
real:[3] previsão:[3]
real:[4] previsão:[4]
Accuracy: 0.975

acertou:9752 falhou:248
Accuracy: 0.975

acertou:9752 falhou:248
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.97      0.96      0.97      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.98      0.98      0.98       892
           6       0.99      0.97      0.98       958
           7       0.97      0.96      0.96      1028
           8       0.97      0.97      0.97       974
           9       0.97      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

[[ 969    0    2    0    0    0    3    2    4    0]
 [    0 1128    1    2    1    1    1    0    1    0]
 [    6    5  993    3    5    0    2    9    8    1]
 [    0    0    2  993    0    5    0    6    2    2]
 [    1    0    1    0  965    0    4    1    3    7]
 [    3    1    0    4    0  876    3    1    4    0]
 [    8    3    1    1    3    7  933    0    2    0]
 [    1    5   16    7    0    0    0  982    2   15]
 [    6    0    2    5    4    1    1    7  944    4]
 [    8    5    1    1    8    4    0    9    4  969]]
```

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

5. Use the model (1)

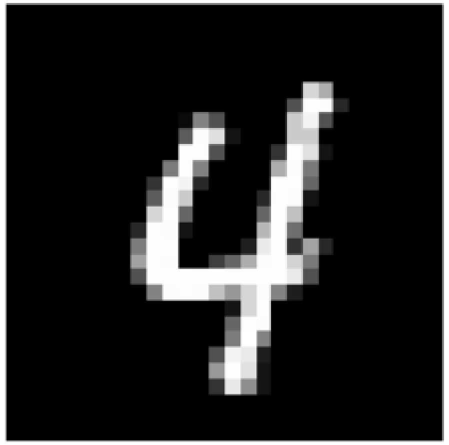
```
def make_prediction(model, img):
    img = img.reshape(1, 1, 28, 28)
    print(img.shape)
    print(img.dtype)
    img = img.to(device) #valves
    prediction = model(img).cpu().detach().numpy()[0].argmax()
    print("predict:",prediction)
    img=img.cpu()
    plt.axis('off')
    plt.grid(b=None)
    plt.imshow(img[0,0], cmap=plt.get_cmap('gray'))
    plt.show()

model= torch.load('CNNModel_1.pth')
imagens, label = next(iter(test_dl))
make_prediction(model,imagens[3])
```

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear
ReLU
Linear
Softmax

5. Use the model (1)

```
def make_prediction(model, img):
    img = img.reshape(1, 1, 28, 28)
    print(img.shape)
    print(img.dtype)
    img = img.to(device) #valves
    prediction = model(img).cpu().detach().numpy()[0].argmax()
    print("predict:",prediction)
    img=img.cpu()
    plt.axis('off')
    plt.grid(b=None)
    plt.imshow(img[0,0], cmap=plt.get_cmap('gray'))
    plt.show()

model= torch.load('CNNModel_1.pth')
imagens, label = next(iter(test_dl))
make_prediction(model,imagens[3])
```

```
torch.Size([1, 1, 28, 28])
torch.float32
predict: 4
```



Exercise 4

- Apply the same process to models 2, 3 and 4, improve and present the best value, detailing the best model

Exercise 4

epochs		
batch size		
lesrning rate		
size splits	test: train:	test: train:
layers + activation functions		
loss function		
optimization function		
accuracy		

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear

2. Define the model (2)

```
class CNNModel_2(Module):
    def __init__(self):
        super(CNNModel_2, self).__init__()
        self.layer1 = Sequential(Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=0),
                                 ReLU(),
                                 MaxPool2d(kernel_size=2)
                                )
        self.layer2 = Sequential(Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=0),
                                 ReLU(),
                                 MaxPool2d(kernel_size=2)
                                )
        self.fc1 = Linear(32 * 5 * 5, 10)
```

Sequential Layer 1(Conv, ReLU, MaxPool)
Sequential Layer 2(Conv, ReLU, MaxPool)
Linear

2. Define the model (2)

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        return out

model = CNNModel_2()
print(summary(model, input_size=(BATCH_SIZE, 1,28,28), verbose=0))
```

2. Define the model (3)

Sequential Layer 1(Conv, BatchNorm, ReLU, MaxPool)
Sequential Layer 2(Conv, BatchNorm, ReLU, MaxPool)
Linear
Dropout
Linear
Linear

```
class CNNModel_3(Module):
    def __init__(self):
        super(CNNModel_3, self).__init__()
        self.layer1 = nn.Sequential(Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1),
                                    BatchNorm2d(32),
                                    ReLU(),
                                    MaxPool2d(kernel_size=2, stride=2)
                                    )
        self.layer2 = nn.Sequential(Conv2d(in_channels=32, out_channels=64, kernel_size=3),
                                    BatchNorm2d(64),
                                    ReLU(),
                                    MaxPool2d(2)
                                    )
        self.fc1 = Linear(in_features=64*6*6, out_features=600)
        self.drop = nn.Dropout2d(0.25)
        self.fc2 = Linear(in_features=600, out_features=120)
        self.fc3 = Linear(in_features=120, out_features=10)
```

2. Define the model (3)

Sequential Layer 1(Conv, BatchNorm, ReLU, MaxPool)
Sequential Layer 2(Conv, BatchNorm, ReLU, MaxPool)
Linear
Dropout
Linear
Linear

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.drop(out)
        out = self.fc2(out)
        out = self.fc3(out)
        return out

model = CNNModel_3()
print(summary(model, input_size=(BATCH_SIZE, 1,28,28), verbose=0))
```

Sequential Layer (Conv, BatchNorm, ReLU, MaxPool, Dropout)
Linear
Linear

2. Define the model (4)

```
class CNNModel_4(Module):
    def __init__(self):
        super(CNNModel_4, self).__init__()
        self.layer1 = Sequential(nn.Conv2d(in_channels=1, out_channels=32, kernel_size=5, padding=0),
                                 BatchNorm2d(32),
                                 ReLU(),
                                 MaxPool2d(2),
                                 Dropout2d(0.2)
                                 )
        self.fc1 = Linear(in_features=32*12*12, out_features=128)
        self.fc2 = Linear(in_features=128, out_features=10)
```

Sequential Layer (Conv, BatchNorm, ReLU, MaxPool, Dropout)
Linear
Linear

2. Define the model (4)

```
    def forward(self, x):
        out = self.layer1(x)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        out = self.fc2(out)
        return out


model = CNNModel_4()
print(summary(model, input_size=(BATCH_SIZE, 1,28,28), verbose=0))
```

Exercise 5

- Apply the same models to fashion-MNIST dataset, improve and present the best value

Exercise 5

epochs		
batch size		
lesrning rate		
size splits	test: train:	test: train:
layers + activation functions		
loss function		
optimization function		
accuracy		

1.1 Visualize the data

```
def output_label(label):
    output_mapping = { 0: "T-shirt/Top", 1: "Trouser", 2: "Pullover",
                       3: "Dress",       4: "Coat",    5: "Sandal",
                       6: "Shirt",       7: "Sneaker", 8: "Bag",
                       9: "Ankle Boot" }
    output_mapping2 = { 0: "0", 1: "1", 2: "2", 3: "3", 4: "4",
                        5: "5", 6: "6", 7: "7", 8: "8", 9: "9"}
    input = (label.item() if type(label) == torch.Tensor else label)
    return output_mapping[input]
```

CNN for multiclass image classification

CIFAR-10 (Canadian Institute For Advanced Research) Dataset

- Image dataset
- Contains 60 000 colour images with 32x32 pixels classified in 10 different classes
- The classes are: planes, cars, birds, cats, deer, frogs, horses, ships and trucks
- There are 6 000 images for each class
- 5 000 images are used for training and 1 000 for testing

0. Prepare the setup

Install pytorch (if needed)

Imports

Constants

```
PATH = './cifar/'
PATH_CLASSES = './cifar/labels.txt'
PATH_TRAIN = './cifar/train'
PATH_TEST = './cifar/test'

BATCH_SIZE = 128
```

Device management (optional)

1. Prepare the data

```
def get_classes(path):
    with open("cifar/labels.txt") as fich_labels:
        labels = fich_labels.read().split()
        classes = dict(zip(labels, list(range(len(labels)))))
    return classes
dic_classes=get_classes(PATH_CLASSES)
print(dic_classes)

def preprocessar(imagem):
    imagem = np.array(imagem)
    cifar_mean = np.array([0.4914, 0.4822, 0.4465]).reshape(1,1,-1)
    cifar_std  = np.array([0.2023, 0.1994, 0.2010]).reshape(1,1,-1)
    imagem = (imagem - cifar_mean) / cifar_std
    xmax, xmin = imagem.max(), imagem.min()
    imagem = (imagem - xmin)/(xmax - xmin)
    imagem = imagem.transpose(2,1,0)
    return imagem
```

1. Prepare the data

```
def get_classes(path):
    with open("cifar/labels.txt") as fich_labels:
        labels = fich_labels.read().split()
        classes = dict(zip(labels, list(range(len(labels)))))
    return classes
dic_classes=get_classes(PATH_CLASSES)
print(dic_classes)
```

```
{'airplane': 0, 'automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4, 'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}
```

```
def preprocessar(imagem):
    imagem = np.array(imagem)
    cifar_mean = np.array([0.4914, 0.4822, 0.4465]).reshape(1,1,-1)
    cifar_std  = np.array([0.2023, 0.1994, 0.2010]).reshape(1,1,-1)
    imagem = (imagem - cifar_mean) / cifar_std
    xmax, xmin = imagem.max(), imagem.min()
    imagem = (imagem - xmin)/(xmax - xmin)
    imagem = imagem.transpose(2,1,0)
    return imagem
```

1. Prepare the data

```
class Cifar10Dataset(Dataset):
    def __init__(self, path, mun_imagens = 0, transforms=None):
        files = os.listdir(path)
        files = [os.path.join(path,f) for f in files]
        if mun_imagens == 0:
            mun_imagens = len(files)
        self.mun_imagens = mun_imagens
        self.files = random.sample(files, self.mun_imagens)
        self.transforms = transforms

    def __len__(self):
        return self.mun_imagens
```

```
def __getitem__(self, idx):
    fich_imagem = self.files[idx]
    imagem = Image.open(fich_imagem)
    imagem = preprocessar(imagem)
    label_classe = fich_imagem[:-4].split("_")[-1]
    label = dic_classes[label_classe]
    imagem = imagem.astype(np.float32)
    if self.transforms:
        imagem = self.transforms(imagem)
    return imagem, label
```

1. Prepare the data

```
def prepare_data_loaders(path_train, path_test):
    dataset_train = Cifar10Dataset(path_train,transforms=None)
    dataset_test = Cifar10Dataset(path_test,transforms=None)

    train_size = int(0.8 * len(dataset_train))
    val_size = len(dataset_train) - train_size
    train, validation = random_split(dataset_train, [train_size, val_size],
                                                                generator=torch.Generator().manual_seed(42))
    train_dl = DataLoader(train, batch_size=BATCH_SIZE, shuffle=True)
    val_dl = DataLoader(validation, batch_size=BATCH_SIZE, shuffle=True)
    test_dl = DataLoader(dataset_test, batch_size=BATCH_SIZE, shuffle=True)
    train_dl_all = DataLoader(train, batch_size=len(train), shuffle=True)
    val_dl_all = DataLoader(validation, batch_size=len(validation), shuffle=True)
    test_dl_all = DataLoader(dataset_test, batch_size=len(dataset_test), shuffle=True)
    return train_dl, val_dl, test_dl, train_dl_all, val_dl_all, test_dl_all

train_dl, val_dl, test_dl, train_dl_all, val_dl_all, test_dl_all = prepare_data_loaders(PATH_TRAIN, PATH_TEST)
```

1.1 Visualize the data

```
def output_label(label,mapping='label'):
    if mapping == 'ext':
        output_mapping = { 0:"zero", 1:"um", 2:"dois", 3:"tres", 4:"quatro", 5:"cinco", 6:"seis", 
7:"sete",                          8:"oito", 9:"nove" }
    elif mapping == 'ext2':
        output_mapping = { "0":"zero", "1":"um", "2":"dois", "3":"tres", "4":"quatro", "5":"cinco", 
                          "6":"seis", "7":"sete", "8":"oito", "9":"nove" }
    else:
        output_mapping = { 0: "0", 1: "1", 2: "2", 3: "3", 4: "4", 5: "5", 6: "6", 7: "7", 8: "8", 9: "9"}
    input = (label.item() if type(label) == torch.Tensor else label)
    return output_mapping[input]
```

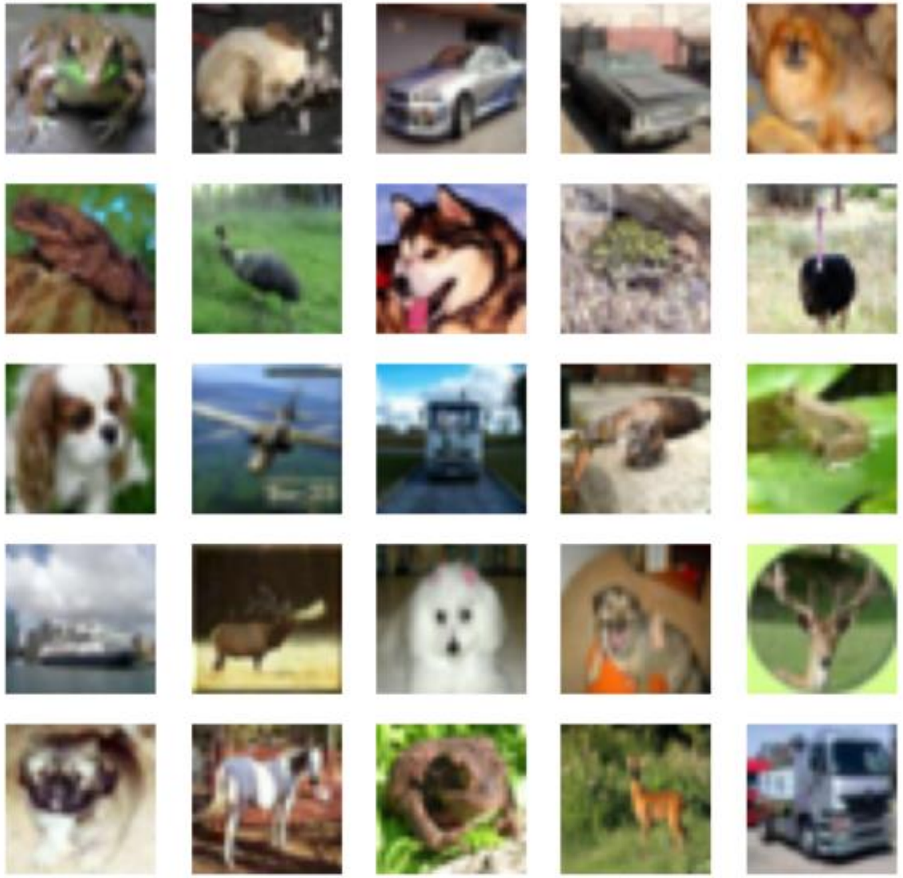
1.1 Visualize the data

```
from IPython.display import display
def visualize_data(path):
    ...

def visualize_dataset(train_dl, test_dl, dataset_train, dataset_test):
    ...

visualize_dataset(train_dl, test_dl, train_dl_all, test_dl_all)

def visualize_images(dl):
    ...

visualize_images(train_dl)
```

1.1 Visualize the data

```
Quantidade de casos de Treino:40000
Quantidade de casos de Validação:10000
Quantidade de casos de Teste:10000
Shape tensor batch casos treino, input: torch.Size([128, 3, 32, 32]), output: torch.Size([128])
Shape tensor batch casos validação, input: torch.Size([128, 3, 32, 32]), output: torch.Size([128])
Shape tensor batch casos test, input: torch.Size([128, 3, 32, 32]), output: torch.Size([128])
Valor maximo:1.0 Valor mínimo:0.0
Valor maximo:1.0 Valor mínimo:0.0
tensor([8, 8, 7, 5, 0, 3, 6, 9, 5, 7, 8, 0, 5, 5, 0, 5, 0, 2, 1, 1, 3, 7, 7, 6,
        9, 5, 3, 0, 2, 6, 5, 1, 5, 1, 8, 1, 7, 8, 9, 4, 8, 3, 6, 0, 7, 8, 1, 1,
        8, 6, 5, 0, 7, 4, 6, 6, 3, 4, 9, 6, 6, 3, 4, 5, 5, 6, 2, 1, 1, 2, 5, 2,
        7, 9, 0, 8, 7, 2, 3, 0, 8, 4, 7, 4, 5, 9, 5, 9, 3, 4, 6, 4, 4, 0, 9, 9,
        6, 0, 8, 8, 1, 0, 4, 8, 6, 7, 7, 1, 9, 2, 5, 5, 3, 7, 7, 9, 6, 0, 4, 2,
        8, 2, 6, 0, 8, 3, 4, 6])
```

```
torch.Size([128, 3, 32, 32])
```



1.2 Verify the dataset balancing

2. Define the model (residual)

Conv Layer 1
Batch Norm 1
Conv Layer 2
Batch Norm 2
Sequential

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=out_channels,kernel_size=(3, 3),
        stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(in_channels=out_channels, out_channels=out_channels,kernel_size=(3, 3),
        stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_channels,kernel_size=(1, 1),
                  stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )
```

2. Define the model (residual)

Conv Layer 1
Batch Norm 1
Conv Layer 2
Batch Norm 2
Sequential

```
    def forward(self, x):
        out = nn.ReLU()(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = nn.ReLU()(out)
        return out

class ResNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=(3, 3),stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.block1 = self._create_block(64, 64, stride=1)
        self.block2 = self._create_block(64, 128, stride=2)
        self.block3 = self._create_block(128, 256, stride=2)
        self.block4 = self._create_block(256, 512, stride=2)
        self.linear = nn.Linear(512, num_classes)
```

2. Define the model (residual)

Conv Layer 1
Batch Norm 1
Conv Layer 2
Batch Norm 2
Sequential

```
def _create_block(self, in_channels, out_channels, stride):
    return nn.Sequential(
        ResidualBlock(in_channels, out_channels, stride),
        ResidualBlock(out_channels, out_channels, 1)
    )

def forward(self, x):
    out = nn.ReLU()(self.bn1(self.conv1(x)))
    out = self.block1(out)
    out = self.block2(out)
    out = self.block3(out)
    out = self.block4(out)
    out = nn.AvgPool2d(4)(out)
    out = out.view(out.size(0), -1)
    out = self.linear(out)
    return out)
```

```
model = ResNet()
print(summary(model, input_size=(BATCH_SIZE, 3,32,32), verbose=0))
```

Conv Layer 1
Batch Norm 1
Conv Layer 2
Batch Norm 2
Sequential

2. Define the model (residual)

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [128, 64, 32, 32]         1,728
├─BatchNorm2d: 1-2                       [128, 64, 32, 32]         128
├─Sequential: 1-3                        [128, 64, 32, 32]         --
│    └─ResidualBlock: 2-1                [128, 64, 32, 32]         --
│    │    └─Conv2d: 3-1                  [128, 64, 32, 32]         36,864
│    │    └─BatchNorm2d: 3-2             [128, 64, 32, 32]         128
│    │    └─Conv2d: 3-3                  [128, 64, 32, 32]         36,864
│    │    └─BatchNorm2d: 3-4             [128, 64, 32, 32]         128
│    │    └─Sequential: 3-5              [128, 64, 32, 32]         --
│    └─ResidualBlock: 2-2                [128, 64, 32, 32]         --
│    │    └─Conv2d: 3-6                  [128, 64, 32, 32]         36,864
│    │    └─BatchNorm2d: 3-7             [128, 64, 32, 32]         128
│    │    └─Conv2d: 3-8                  [128, 64, 32, 32]         36,864
│    │    └─BatchNorm2d: 3-9             [128, 64, 32, 32]         128
│    │    └─Sequential: 3-10             [128, 64, 32, 32]         --
├─Sequential: 1-4                        [128, 128, 16, 16]        --
│    └─ResidualBlock: 2-3                [128, 128, 16, 16]        --
│    │    └─Conv2d: 3-11                 [128, 128, 16, 16]        73,728
│    │    └─BatchNorm2d: 3-12            [128, 128, 16, 16]        256
│    │    └─Conv2d: 3-13                 [128, 128, 16, 16]        147,456
│    │    └─BatchNorm2d: 3-14            [128, 128, 16, 16]        256
│    │    └─Sequential: 3-15             [128, 128, 16, 16]        8,448
│    └─ResidualBlock: 2-4                [128, 128, 16, 16]        --
│    │    └─Conv2d: 3-16                 [128, 128, 16, 16]        147,456
│    │    └─BatchNorm2d: 3-17            [128, 128, 16, 16]        256
│    │    └─Conv2d: 3-18                 [128, 128, 16, 16]        147,456
│    │    └─BatchNorm2d: 3-19            [128, 128, 16, 16]        256
│    │    └─Sequential: 3-20             [128, 128, 16, 16]        --
```

```
├─Sequential: 1-5                        [128, 256, 8, 8]          --
│    └─ResidualBlock: 2-5                [128, 256, 8, 8]          --
│    │    └─Conv2d: 3-21                 [128, 256, 8, 8]          294,912
│    │    └─BatchNorm2d: 3-22            [128, 256, 8, 8]          512
│    │    └─Conv2d: 3-23                 [128, 256, 8, 8]          589,824
│    │    └─BatchNorm2d: 3-24            [128, 256, 8, 8]          512
│    │    └─Sequential: 3-25             [128, 256, 8, 8]          33,280
│    └─ResidualBlock: 2-6                [128, 256, 8, 8]          --
│    │    └─Conv2d: 3-26                 [128, 256, 8, 8]          589,824
│    │    └─BatchNorm2d: 3-27            [128, 256, 8, 8]          512
│    │    └─Conv2d: 3-28                 [128, 256, 8, 8]          589,824
│    │    └─BatchNorm2d: 3-29            [128, 256, 8, 8]          512
│    │    └─Sequential: 3-30             [128, 256, 8, 8]          --
├─Sequential: 1-6                        [128, 512, 4, 4]          --
│    └─ResidualBlock: 2-7                [128, 512, 4, 4]          --
│    │    └─Conv2d: 3-31                 [128, 512, 4, 4]          1,179,648
│    │    └─BatchNorm2d: 3-32            [128, 512, 4, 4]          1,024
│    │    └─Conv2d: 3-33                 [128, 512, 4, 4]          2,359,296
│    │    └─BatchNorm2d: 3-34            [128, 512, 4, 4]          1,024
│    │    └─Sequential: 3-35             [128, 512, 4, 4]          132,096
│    └─ResidualBlock: 2-8                [128, 512, 4, 4]          --
│    │    └─Conv2d: 3-36                 [128, 512, 4, 4]          2,359,296
│    │    └─BatchNorm2d: 3-37            [128, 512, 4, 4]          1,024
│    │    └─Conv2d: 3-38                 [128, 512, 4, 4]          2,359,296
│    │    └─BatchNorm2d: 3-39            [128, 512, 4, 4]          1,024
│    │    └─Sequential: 3-40             [128, 512, 4, 4]          --
├─Linear: 1-7                            [128, 10]                 5,130
```

```
---------------------------------------------
Total params: 11,173,962
Trainable params: 11,173,962
Non-trainable params: 0
Total mult-adds (T): 1.34
=============================================
Input size (MB): 1.57
Forward/backward pass size (MB): 1258.30
Params size (MB): 44.70
Estimated Total Size (MB): 1304.57
=============================================
```

3. Train the model (residual)


```
def train_model(h5_file, train_dl, val_dl, model, criterion, optimizer):
    ...
```

For ResNet model:

```
model = ResNet()
print(summary(model, input_size=(BATCH_SIZE, 3,32,32), verbose=0))
EPOCHS = 30
LEARNING_RATE = 0.001
criterion = CrossEntropyLoss()
optimizer = SGD(model.parameters(), lr=LEARNING_RATE)
starttime = time.perf_counter()
train_model('CNNModel_cifar_Resnet.pth', train_dl, val_dl, model, criterion, optimizer)
endtime = time.perf_counter()
print(f"Tempo gasto: {endtime - starttime} segundos")
```

Conv Layer 1
Batch Norm 1
Conv Layer 2
Batch Norm 2
Sequential

4. Evaluate the model (residual)


```
def evaluate_model(test_dl, model):
    ...
def display_predictions(actual_values, predictions):
    ...
def display_confusion_matrix(cm,list_classes):
    ...
actual_values, predictions = evaluate_model(test_dl_all, model)

model= torch.load('CNNModel_cifar_Resnet.pth')
actual_values, predictions = evaluate_model(test_dl_all, model)
display_predictions(actual_values, predictions )
print(classification_report(actual_values, predictions))
cr =classification_report(actual_values, predictions, output_dict=True)
list_classes=[output_label(n,'ext2') for n in list(cr.keys())[0:10] ]
cm = confusion_matrix(actual_values, predictions)
print (cm)
display_confusion_matrix(cm,list_classes)
```

Conv Layer 1
Batch Norm 1
Conv Layer 2
Batch Norm 2
Sequential

5. Use the model (residual)

```
def make_prediction(model, img):
    ...

model= torch.load('CNNModel_cifar_Resnet.pth')
imagens, label = next(iter(test_dl))
make_prediction(model,imagens[3])
```

Exercise 6

- Apply the same process to models 1, 2, 3 and 4, improve and present the best value, detailing the best model

Exercise 6

epochs		
batch size		
lesrning rate		
size splits	test: train:	test: train:
layers + activation functions		
loss function		
optimization function		
accuracy		