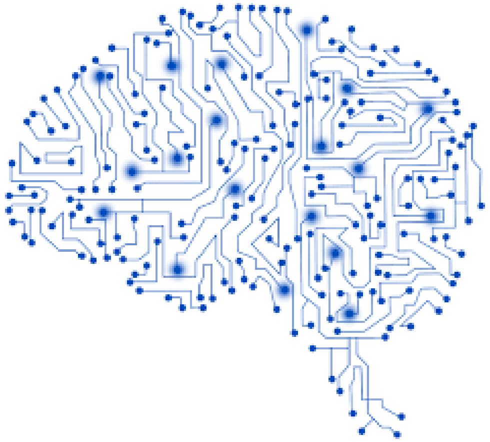
University of Minho
School of Engineering



Aprendizagem Profunda

Deep Neural Network – CNN Multiclass

AP @ MEI/1º ano – 2º Semestre

Victor Alves

Part III

Hands On

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]
```

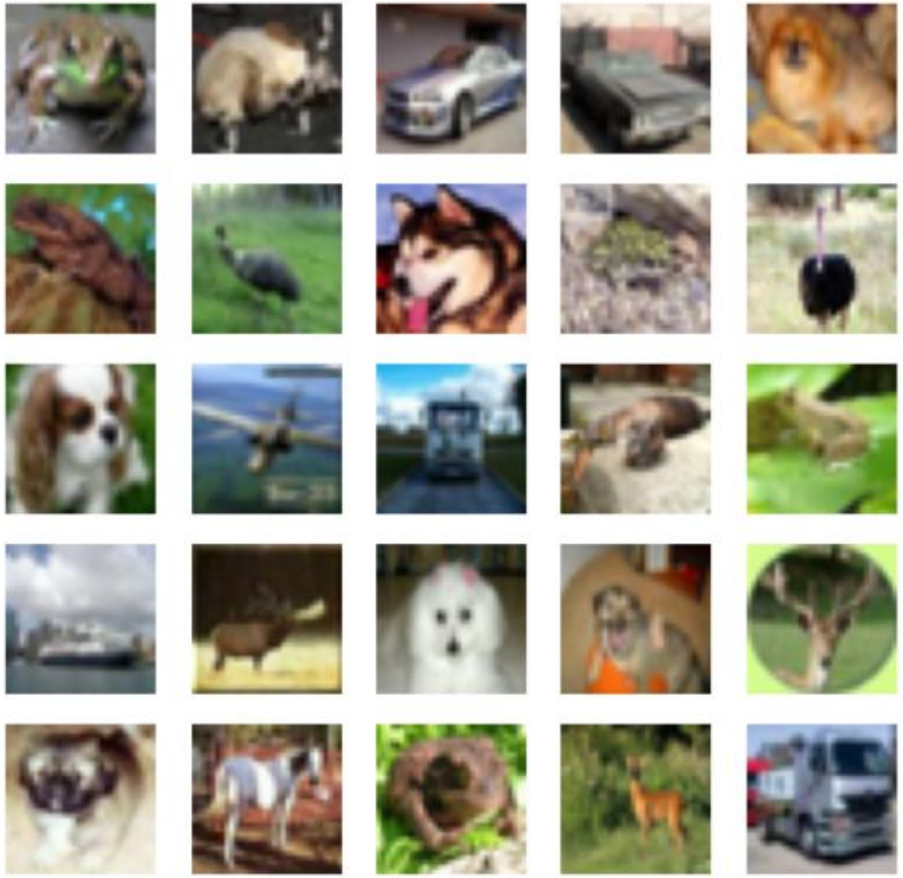
1.1 Visualize the data

```
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

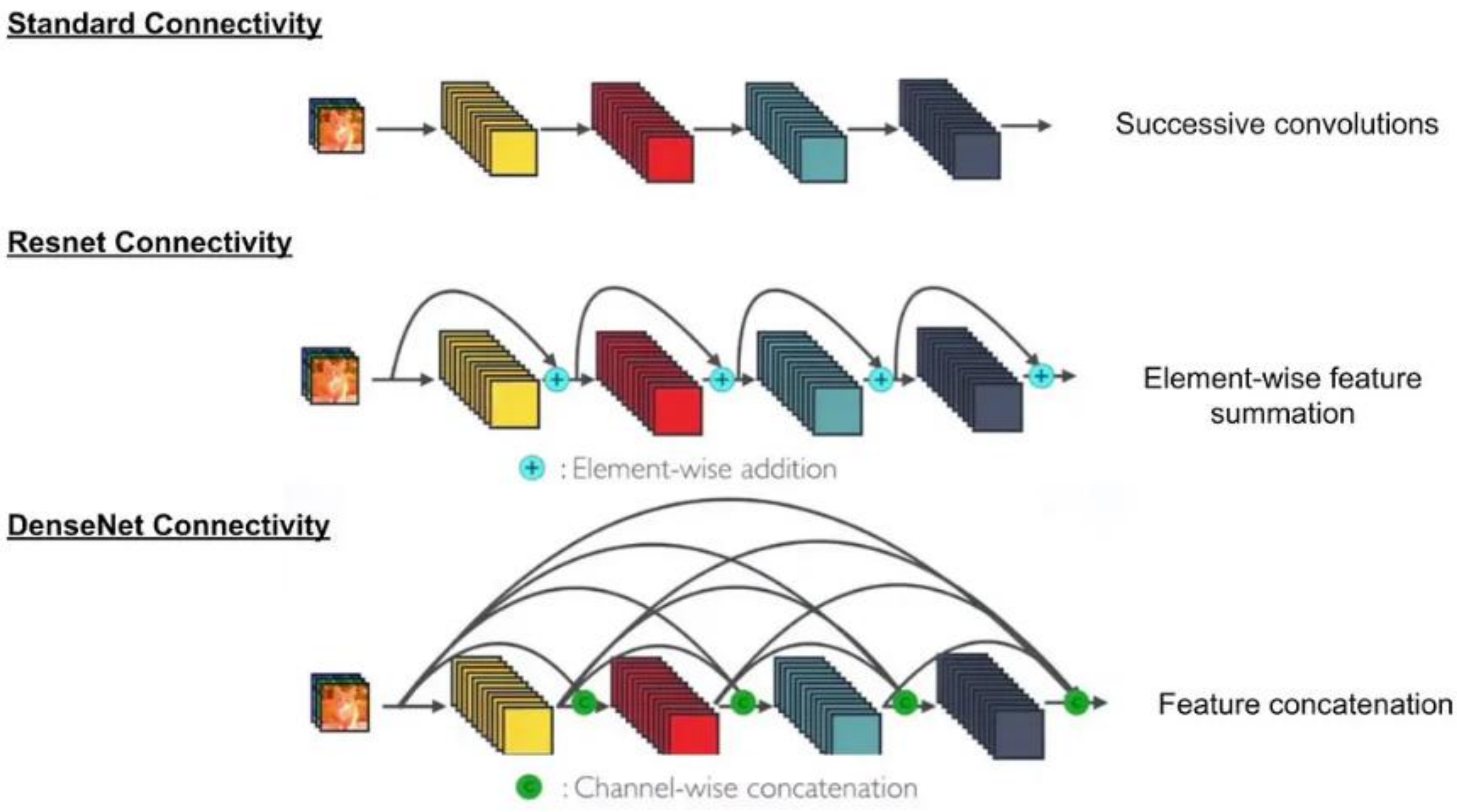
```
Quantidade de casos de Treino:40000
Quantidade de casos de Validação:10000
Quantidade de casos de Teste:10000
Shape tensor batch casos treino, input: torch.Size([128, 3, 32, 32]), output: torch.Size([128])
Shape tensor batch casos validação, input: torch.Size([128, 3, 32, 32]), output: torch.Size([128])
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. Standard, ResNet and DenseNet Connectivity



Connection Patterns of Vanilla CNN, ResNet and DenseNet

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))
```

2. Define the model (residual)

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

```
==========================================================================================
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
==============================================
```

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

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		