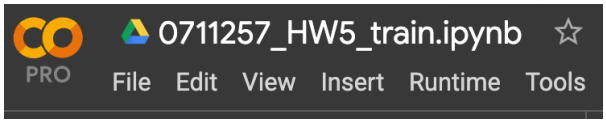
Intro. to Machine Learning

Model weights can be downloaded in:
https://drive.google.com/drive/folders/1DF32ub0L3HoYUu32ah29FnfcvYJ1zdlH?usp=sharing
And edit the path variables for inference.

```
CAPTCHA_DICT = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                'a', 'c', 'd', 'e', 'f', 'h','j', 'k', 'm', 'n', 'p', 'r', 's', 't','v','w','x','y' ]
TEST_PATH = "/content/test"
model1_path = "/content/model1.pth"
model2_path = "/content/model2.pth"
model3_path = "/content/model3.pth"
device = "cuda"
```

Environment:

Do all the things on Colab.



Task1:

Data preprocessing:

Blur, take the average of RGB, normalize.

```
img = cv2.medianBlur(img, 3)
img = np.mean(img, axis=2)
img = torch.FloatTensor((img - 128) / 128)
```

Model architecture:

A simple 4 layers CNN.

```
self.conv1 = nn.Sequential(
nn.Conv2d(in_channels = 1, out_channels = 16, kernel_size = 3, stride = 1, padding = 1,),
nn.BatchNorm2d(16),
nn.ReLU(),# (16, 72, 72)
nn.MaxPool2d(kernel_size = 2)# (16, 36, 36)
)
self.conv2 = nn.Sequential(
nn.Conv2d(16, 32, 3, 1, 1),# (32, 36, 36)
nn.BatchNorm2d(32),
nn.ReLU(),# (32,36,36)
nn.MaxPool2d(2)# (32, 18, 18)
)
self.conv3 = nn.Sequential(
nn.Conv2d(32, 64, 3, 1, 1),# (64, 18, 18)
nn.BatchNorm2d(64),
nn.ReLU(),# (64,18,18)
nn.MaxPool2d(2)# (64, 9, 9)
)
self.conv4 = nn.Sequential(
nn.Conv2d(64, 128, 3, 1, 1),# (128, 9, 9)
nn.BatchNorm2d(128),
nn.ReLU(),# (128,9,9)
)
self.out = nn.Sequential(
  nn.Linear(128*9*9, 1024),
  nn.ReLU(),
  nn.Dropout(),
  nn.Linear(1024, 256),
  nn.ReLU(),
  nn.Dropout(),
  nn.Linear( 256, 10)
)
```

Hyperparameters:

Batch size: 32

Train for about 150 epochs, set the learning rate to 1e-4 when the loss < 0.5.

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.CrossEntropyLoss()
```

Task2:

Data preprocessing:

Encode the label to one hot code for task2 and task3.

```
def encoding(label):
    onehot_code = np.zeros(len(CAPTCHA_DICT) * len(label), dtype=float)
    for i, char in enumerate(label):
        index = i * len(CAPTCHA_DICT) + CAPTCHA_DICT.index(char)
        onehot_code[index] = 1.0
    return onehot_code
```

Resize the images to 108*108 first.

```
  img = cv2.resize(img, (108, 108), interpolation=cv2.INTER_LANCZOS4)
  img = cv2.medianBlur(img, 3)
  img = np.mean(img, axis=2)
  img = torch.FloatTensor((img - 128) / 128)
```

Model architecture:

Train on resnet18 with pretrained weight, modifying the output of the last layer and the input channel of the first layer to fit the captcha data.

```
model2=models.resnet18(pretrained=True)
fc_features = model2.fc.in_features
model2.fc = nn.Linear(fc_features, len(CAPTCHA_DICT)*2)
model2.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
```

Hyperparameters:

Batch size: 36
Train for about 75 epochs, set the learning rate to 1e-4 when the loss < 0.05, and train further on the validation data for about 35 epochs.

```
optimizer = torch.optim.Adam(model2.parameters(), lr=1e-3)
loss_fn = nn.MultiLabelSoftMarginLoss()
```

Task3:

Data preprocessing:

Resize the images to 144*108 first, no median blur in this task.

```
img = cv2.resize(img, (144, 108), interpolation=cv2.INTER_LANCZOS4)
img = np.mean(img, axis=2)
img = torch.FloatTensor((img - 128) / 128)
```

Model architecture:

Train on resnet18 with pretrained weight, modifying the output of the last layer and the input channel of the first layer to fit the captcha data.

```
model3=models.resnet18(pretrained=True)
fc_features = model3.fc.in_features
model3.fc = nn.Linear(fc_features, len(CAPTCHA_DICT)*4)
model3.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
```

Hyperparameters:

Batch size: 72
Train for about 125 epochs, set the learning rate to 1e-4 when the loss < 0.02, and train further on the validation data for about 30 epochs.

```
optimizer = torch.optim.Adam(model3.parameters(), lr=1e-3)
loss_fn = nn.MultiLabelSoftMarginLoss()
```