Reconhecimento automático de placas usando o *Deep Learning*

Passo 1: Estudo do ALPR

O ALPR é um sistema e leitor que foi popularizado pelos vários casos de uso na vida real. Ele é responsável por reconhecer e identificar placas de veículos de forma automática, utilizando uma inteligência artificial treinada. Ele compara digitalmente as informações constantes na placa com a base de dados do sistema, garantindo uma ágil conferência e verificação em tempo real.

Pode ser integrado em diversos sistemas para controle, segurança, geração de estatísticas, rastreabilidade e até gerenciamento de estacionamentos e pagamentos de pedágios.

O mesmo funciona através de um processo do qual se consiste em 2 etapas:

i) Detecção de placas de veículos;

ii) Leitura dos caracteres da placa.

Um exemplo de funcionamento é imaginar a seguinte situação, um motorista ultrapassa uma câmera com uma placa com limite de 80km/h, porém o limite de velocidade permitido 70km/h. Depois de algumas semanas o motorista recebe uma multa com alguns comprovantes que ele foi o autor do incidente.

Para que essa situação possa ocorrer sem que seja feita manualmente com cada imagens e vídeos, o ALPR irá extrair os números das placas com reconhecimento óptico de caracteres.

O reconhecimento automático da placa pode ser usado para armazenar todas as imagens capturadas por câmeras e também o texto encontrado nas mesmas, podendo até mesmo realizar ajustes para que se possa armazenar também uma fotografia do motorista. Esses sistemas, em grande parte, utilizam iluminação infravermelha para que as capturas de imagem possam ser feitas durante qualquer horário do dia.

Passo 2: Código em funcionamento

Podemos dividir o funcionamento do código em um processo de duas etapas:

* Detecção e Reconhecimento.

Para dar início, é necessária primeiramente a instalação do framework Darknet.

Após isso, devemos treinar o modelo em um dataset composto por imagens de veículos. Antes de começar o treinamento, devemos modificar o arquivo de configuração .cfg, mais especificamente nos parâmetros de batch size, subdivision e classes.

Agora, para que o modelo acesse essas informações, são criados dois arquivos, um dos quais contém as informações de treino, teste e classes. Esse arquivo será chamado de “obj.data”, enquanto o outro será “obj.names”, que possui os nomes das classes.

O próximo passo é baixar os pesos pré-treinados em YOLOv4, que é um algoritmo de detecção de objetos.

A partir de agora, damos início ao treinamento, onde incluímos os argumentos os arquivos “obj.data”, de configuração e os pesos treinados em YOLOv4.

O detector já está treinado, e é hora de colocarmos ele para funcionar. Utilizamos a função “yolo\_det()” para detectar os formatos retangulares das placas capturadas das imagens.

Primeiro, fazemos o pré-processamento da imagem de entrada, depois, passamos a imagem para o detector e guardamos o que foi detectado. Após isso, são utilizadas as funções Darknet em cima das detecções, calculado o tempo gasto e taxa de quadros por segundo para detecção. Por fim, é feito o redimensionamento do formato da placa para que se adeque à resolução da imagem.

Definimos um “resize\_bbox()” para fazer esse redimensionamento.

Terminado este processo, partimos agora para o reconhecimento de texto, para que possamos ler as placas detectadas.

Para isso, vamos utilizar outro framework chamado PaddleOCR, que é útil para facilitar o processo de aplicar e treinar diferentes modelos com poucas linhas de código.

Instalamos as ferramentas e suas dependências, depois, inicializamos o OCR, que exigirá alguns argumentos, como a linguagem de reconhecimento, especificação do algoritmo de detecção de texto utilizado e também do algoritmo de reconhecimento.

Passamos uma linha de código aonde o “cr\_img” é a imagem repassada para o OCR. CLS e DET são os argumentos que são setados como falsos.

Agora que o detector de placas está totalmente treinado e o OCR está finalizado, é hora de colocarmos os dois para funcionar juntos. Para isso, criaremos algumas funções para acessar todas as funcionalidades de uma só vez.

Criamos uma função chamada crop() que será responsável por cortar a imagem utilizando suas coordenadas como parâmetros. Depois disso, criamos uma função final chamada “test\_img()”, que irá realizar a detecção, corte, OCR e demais funções.

Antes disso, inicializaremos algumas variáveis visuais para cores e fontes. Depois, carregamos a rede Darknet e suas classes, fazemos a leitura da imagem com a detecção YOLOv4, extraímos/recortamos a placa e aplicamos o OCR.

Por fim, fazemos as previsões com OpenCV e retornamos a imagem de saída.