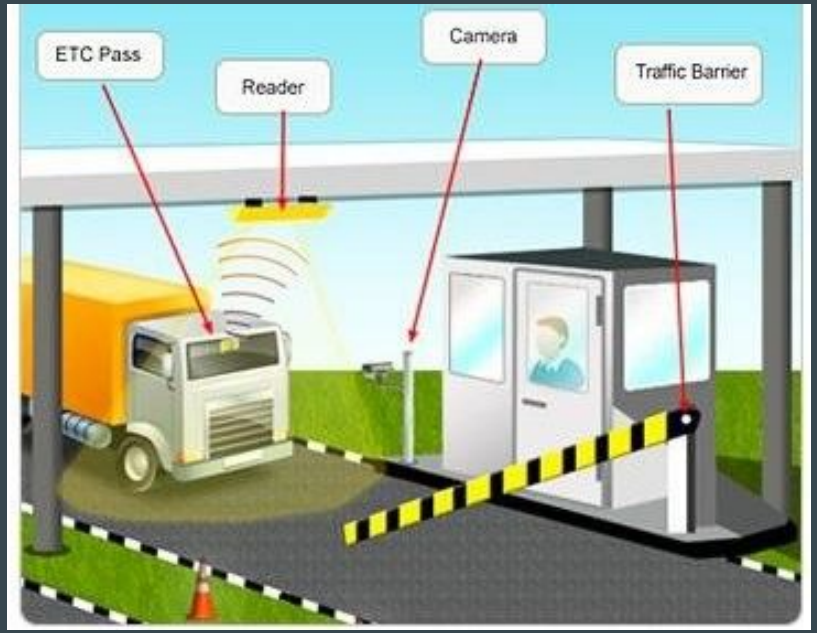
Identifying Car Brands in Noisy Images Using Convolutional Neural Networks

By: Rohit Amarnath, Mugdhesh Pandkar, Aravind Kumar

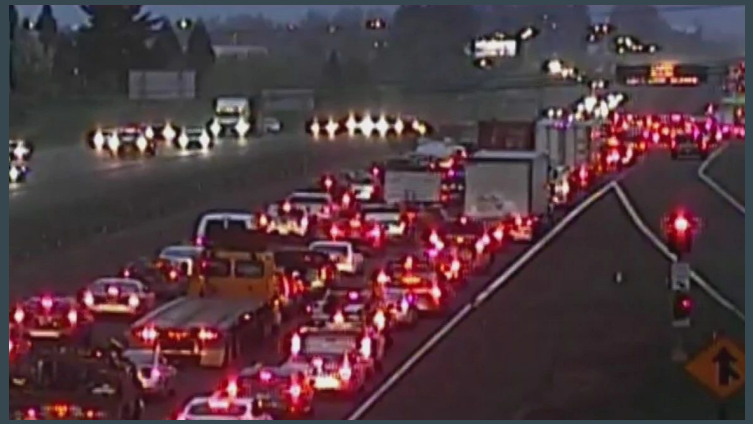
Problem Overview

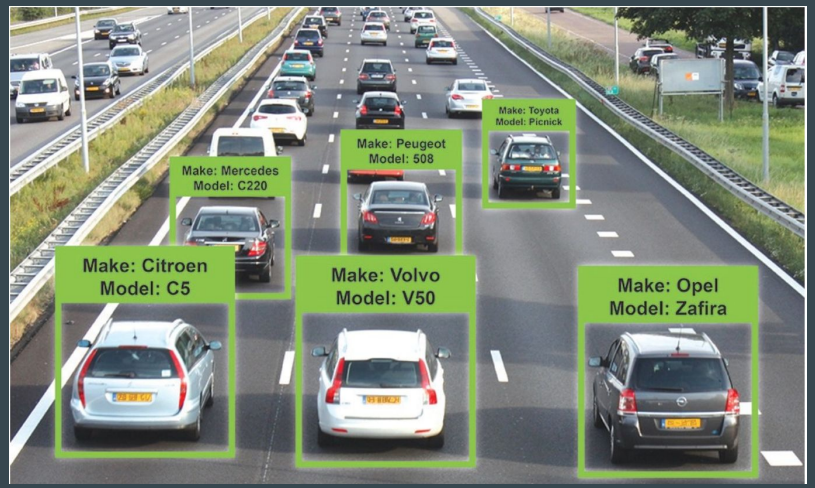
- Exponential production of cars has resulted in the need for better vehicle classification systems
  - E.g. automated highway toll collection, traffic flow control systems, perception in self-driving vehicles
- Classification systems must improve in terms of their ability to identify a car based off of an image
  - Image can have large degree of noise and/or blur



Project Overview

- Most existing models have been trained on clear car images
- Not representative of real world conditions
  - E.g. inclement weather, rain, motion blur
- Our objective: classify car brands in noisy and/or blurry images using convolutional neural networks
  - Project uses AutoEncoders & ResNet architecture



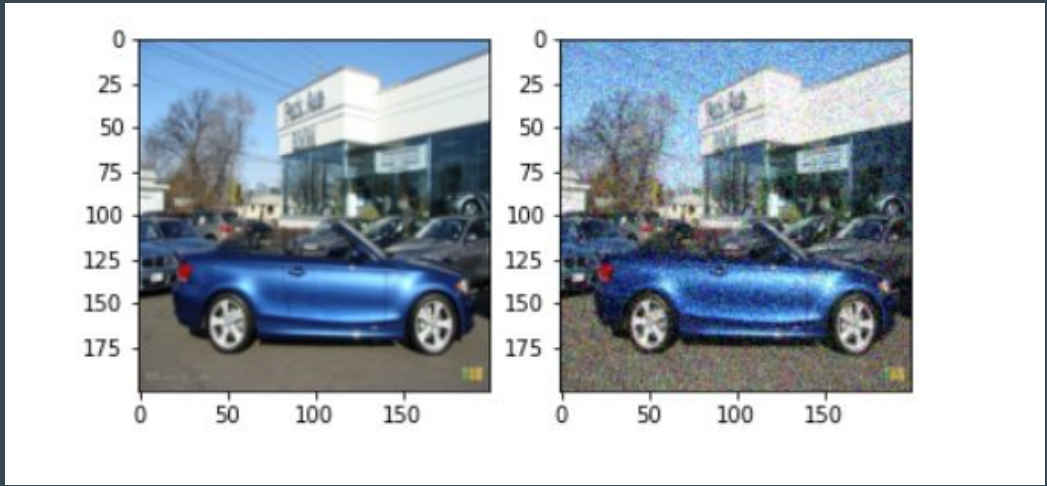


Datasets Used

- Stanford Cars Dataset
  - Consists of 16,185 images and 196 classes of cars
  - Each class refers to a car's make, model, and year
  - E.g. Honda Accord Sedan 2012

Stages from Input to Output: Noise Function

- Input: Clear car images from Stanford cars dataset
- Not representative of real world conditions
- We implemented an algorithm to add noise to these clear car images to better simulate real world conditions



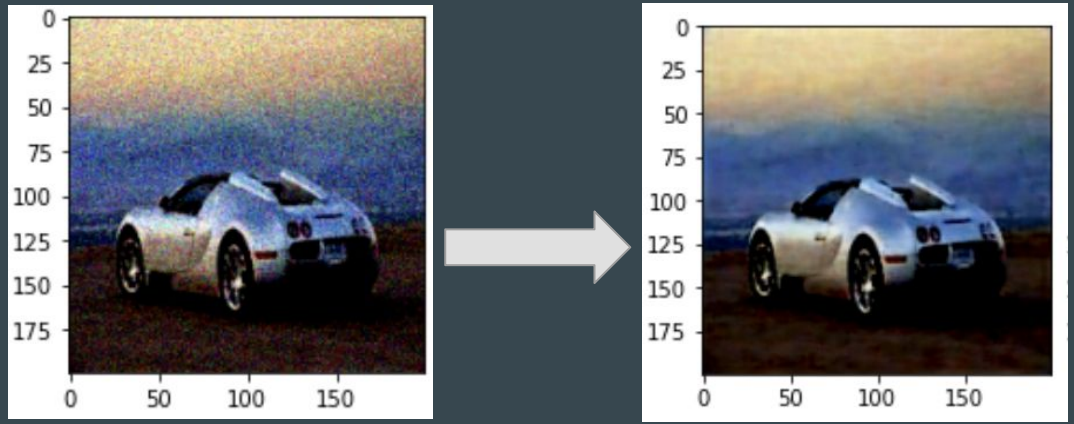
Noise Function

```
[ ]  def add_noise(image_batch_inputs, noise_factor = 0.12):
       noisy_batch = image_batch_inputs + torch.randn_like(image_batch_inputs) * noise_factor
       noisy_batch = torch.clip(noisy_batch,0.,1.)
       return noisy_batch
```

First step: adding noise to clear car images from dataset to use as input

Stages from Input to Output: Denoising Model

- Take noisy images from prior step as input
- Defines encoder and decoder comprised of 3 Conv2D and 3 ReLU layers.
- The model accepts a noisy image as input and tries to remove much of the noise



AutoEncoder Architecture

```
[ ]  class AutoEncoder(nn.Module):
        def __init__(self, device):
            super().__init__()
            self.encoder = nn.Sequential(
              nn.Conv2d(3, 16, kernel_size = 5),
              nn.ReLU(True),
              nn.Conv2d(16, 32, kernel_size = 5),
              nn.ReLU(True),
              nn.Conv2d(32, 64, kernel_size = 5),
              nn.ReLU(True)
            )

            self.decoder = nn.Sequential(
                nn.ConvTranspose2d(64, 32, kernel_size = 5),
                nn.ReLU(True),
                nn.ConvTranspose2d(32, 16, kernel_size = 5),
                nn.ReLU(True),
                nn.ConvTranspose2d(16, 3, kernel_size = 5),
                nn.ReLU(True)
            )
            self.device=device
        def forward(self, x):
          x = self.encoder(x)
          x = self.decoder(x)
          return x
```

Hyperparameters

```
[ ]  weight_decay = 1e-5
     learning_rate = 1e-3
```

Stages from Input to Output: Training Algorithm and Loss

- We then trained our model with 15 epochs
- Recorded loss in each epoch
- Loss of 0.1560 in last iteration (204) of final epoch

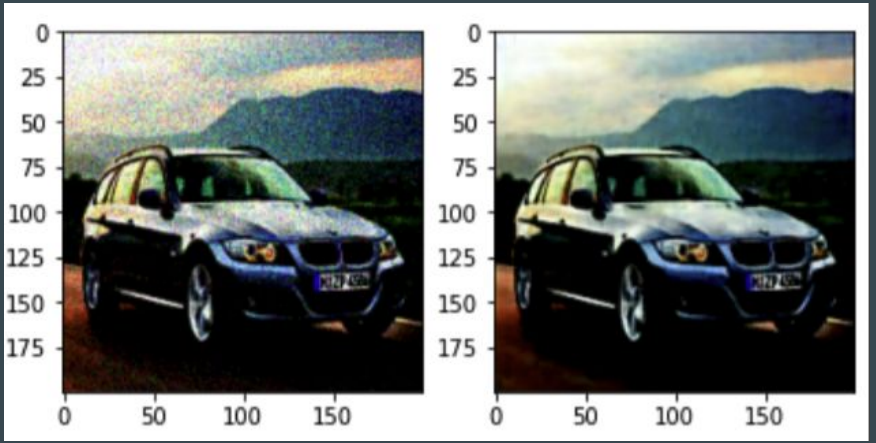
```
[ ]  num_epochs = 15
     iterations_training_loss = []
     epochs_training_loss = []

     for epoch in range(num_epochs):
         print('Epoch [{}/{}]'.format(epoch + 1, num_epochs))
         iteration = 0
         for data in noise_train_loader:
             data = [image.to(device) for image in data]
             iteration = iteration + 1
             img, _ = data
             noisy_images = add_noise(img)
             output = noise_model(noisy_images)
             loss = distance(output, img)
             optimizer.zero_grad()
             loss.backward()
             optimizer.step()
             print('Epoch [{}/{}], Iteration: {}, Loss: {:.4f}'.format(epoch + 1, num_epochs, iteration, loss.item()))
             iterations_training_loss.append(loss.item())
```

```
Epoch [15/15], Iteration: 202, Loss: 0.1777
Epoch [15/15], Iteration: 203, Loss: 0.1978
Epoch [15/15], Iteration: 204, Loss: 0.1560
```

Recorded final loss of 0.1560

Stages from Input to Output: Testing Set and Final Results

- After training the denoising model on the training set, we tested our results on the test set
- Returns noisy input image and denoised output image
- Results show high degree of denoising



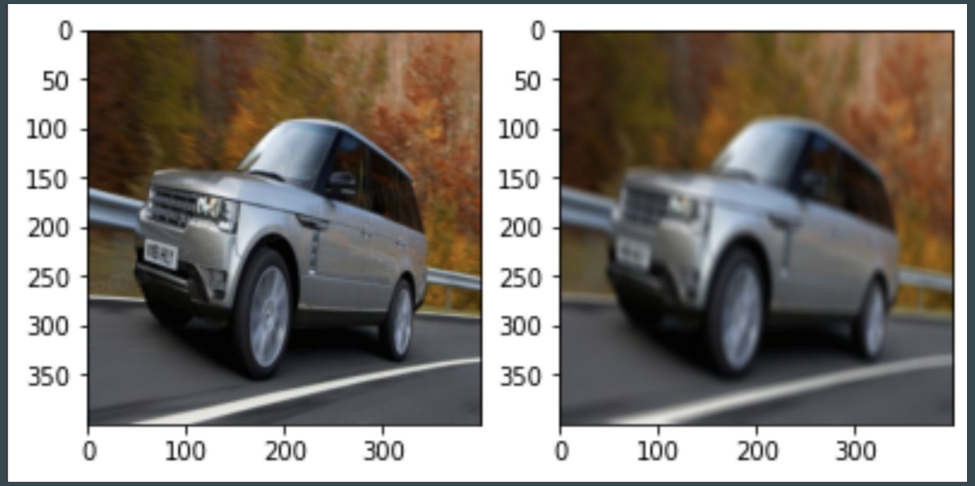
Resnet Architecture

- Used Resnet Architecture to classify cars
- Given an image of a car, we predict it's make, model and year
- Lower confidence but able to predict high level of specificity
- Not possible to build own model & train model
  - ResNet is trained in ImageNet
  - We detached the fully connected layer and added a linear layer allowing, extrapolating it to a 196 class layer


```
resnet_model.fc = nn.Linear(num_features, 196)
resnet_model = resnet_model.to(device)
```

Next steps

- Link denoising algorithm to Resnet architecture to build comprehensive car classifier
- Incorporate blurred images:
  - Design processes to blur and deblur images
  - Integrate with ResNet to classify cars
- Potentially combine noisy and blurred images



Thank You!