# Development of Computer Vision Models for Drivable Region Detection in Snow Occluded Lane Lines


**Parth Kadav, Sachin Sharma, Farhang Motallebi Araghi, and Zachary D. Asher**

## 1 Introduction

Advanced Driver Assistance Systems (ADAS) have the ability to prevent or reduce around 40% of all passenger vehicle incidents [1]. Some examples of ADAS include forward collision warning (FCW), automatic emergency braking (AEB), lane departure warning (LDW), lane-keeping assistance (LKA), and blind-spot warning assistance, among others. Since human error is the leading cause of road accidents [2], ADAS was designed to automate and improve aspects of the driving experience in order to increase road safety and safe driving habits. Lane-keeping systems detect reflective lane markers in front the vehicle and warn the driver via various audible, tactile, and/or visual cues if the vehicle deviates from its lane and no turn signals or steering movements are detected [3]. LDW/LKA systems can reduce head-on and single-vehicle collisions by 53% on highways with higher speed limits (45–75 mph) with visible lane markings, according to a study of 1853 driver injury crashes [4, 5]. 11%–23% of drift-out-of-lane events and 13%–22% of critically to fatally injured drivers could have been prevented if the technology had been implemented at lower operating speeds (5–20 mph), according to [6]. FCW and AEB alone significantly halve front-to-rear crashes [7]. By 2023, it is anticipated that the market for ADAS would be worth more than $30 billion [8] and that ADAS will not be limited to safety but will also enable improvements in vehicle efficiency [9–14].

P. Kadav (✉) · S. Sharma · F. M. Araghi · Z. D. Asher
Department of Mechanical and Aerospace Engineering, Western Michigan University, Kalamazoo, MI, USA
e-mail: parth.kadav@wmich.edu; sachin.sharma@wmich.edu; farhang.motallebiaraghi@wmich.edu; zach.asher@wmich.edu



V. K. Kukkala, S. Pasricha (eds.), *Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*, https://doi.org/10.1007/978-3-031-28016-0_21

Despite the success of ADAS technology, there remains a glaring issue: adverse weather. In the United States, weather-related crashes accounted for 21% (1,235,145) of all recorded crashes, 16% (5376) of crash fatalities, and 19% (418,005) of crash injuries between 2007 and 2016 [15]. Fundamentally, adverse weather conditions can hinder situational awareness and vehicular maneuverability in a variety of ways, depending on the type of adverse weather [15]. It is critical to recognize how various weather conditions can affect the ground transportation infrastructure. A current research problem is to develop strategies for operating ADAS in bad weather. Because there are significant safety implications, the first research gap is to recognize and classify road lanes during inclement weather in order to aid in the location of both the ego vehicle and other vehicles [16]. The difficulty is that inclement weather, such as heavy rain, snow, or fog, reduces the maximum range and signal quality of ADAS sensors, such as cameras, as it obscures the lane markings [16]. This issue has been illustrated with cameras and lidars in particular [17]. According to [4], LDW/LKA could further reduce head-on and single-vehicle collisions on roads with operating speeds of 45–75 mph by 53% only if the roads had visible road markings and "the road surface was not coated by ice or snow." The performance of new sensor technologies is improving, but not enough to address the issue of reliable ADAS operation in inclement weather [9]. To address this research gap, this study concentrates on the snow covered roads to keep the research scope reasonable.

There are only a few significant studies that address the issue of reliable ADAS operation in snowy conditions. The first study created a customized snowy weather dataset and determined the driveable region using semantic segmentation [18]. When assessed on a non-snow dataset, the model's mean Intersection over Union (mIoU) was 80%; when trained on a snowy dataset, mIoU fell to 19%. When both models were combined, mIoU was 83.3%. The model must be improved and strengthened because it analyzes the entire road rather than just the Region of Interest (ROI), which can be computationally costly. The second study used a CNN model with a predefined architecture and sensor fusion between the camera, lidar, and radar [19]. A dataset test showed an increase in driveable region detection (81.35%) and non-driveable region detection (93.85%) after combining data from several sensors. This is an improvement, but it has downsides, the most notable of which is that the method necessitates the use of more sensors, raising the cost and computational power required. Additionally, like the first study, this technique examines the full driveable zone rather than just a ROI [19]. In a third study, "You Only Look Once" (YOLO) was combined with a CNN and Federated Learning (FL) architecture to increase detection in inclement weather [20]. The Canadian Adverse Driving Conditions (CADC) dataset was used to evaluate this method. The average test accuracy of the model used in their study was 82.4%–88.1% . This model is based on the FL technique, which utilizes an edge server. The edge server transmits the initial parameters to the AVs after training a global YOLO CNN model on a publically available dataset. Following that, the AVs utilize these parameters to locally train the model on their own data. Once the local models are trained on each vehicle, they are sent back to the edge server. The training time of the FL approach

is influenced by the number of AVs collecting data, the connection between the edge server and each vehicle, and the processing power of each vehicle. In addition, the vehicle has been fitted with eight cameras, resulting in an increase in price [20]. All of the above mentioned research provides strategies for enhancing the identification of objects and regions in the full driveable environment, but not necessarily the lane information. These studies are computationally and monetarily expensive and rely on several sensors. None of these studies offer precise, implementable driveable region detection for snow-covered roads using a single camera sensor in ADAS systems. Furthermore, custom data acquisition and labeling methods on a custom dataset are not included in these studies. A study addressing these difficulties and discussing unique CNN architectures to improve drivable region prediction with limited data is required.

We devised a computationally efficient, cost-effective, and high-accuracy technique for extracting driveable region information from a single camera, a ubiquitous vehicle sensor, to address the adverse weather research gap for ADAS [17, 18]. Deep Learning (DL) approaches such as Convolutional Neural Network (CNN) have been established as the dominant paradigm in modern computer vision algorithms and applications, as well as in segmentation research. CNNs are a robust method of obtaining semantic segmentation, but are generally computationally intensive when compared to classical ML models. Classic ML models are faster at real-time compute speeds, but they require feature engineering and pre-processing, and they do not serve as an end-to-end solution for identifying the drivable region in snow-covered lane lines, which we know from previous work [21]. To solve this problem, we will investigate DL techniques that need little or no feature engineering. For semantic segmentation, both supervised classical ML models and custom CNN models were created. Then, these methods for detecting tire marks in snow were compared. To broaden the scope of the research, we will build five different CNN architectures for determining the drivable region in snow-occluded lane lines using a single camera sensor.

## 2 Methodology

In this section, we will first discuss the methods we used to collect and prepare the data. The data that has been processed is then used to develop the classical ML models and the Deep Neural Network models

### *2.1 Drive Cycles*

Figure 1 shows the route we chose which consisted of two-lane arterial roads in Kalamazoo that met our criteria for road characteristics. This drive cycle included of roads that are rarely cleaned following winter and are maintained at a much lower

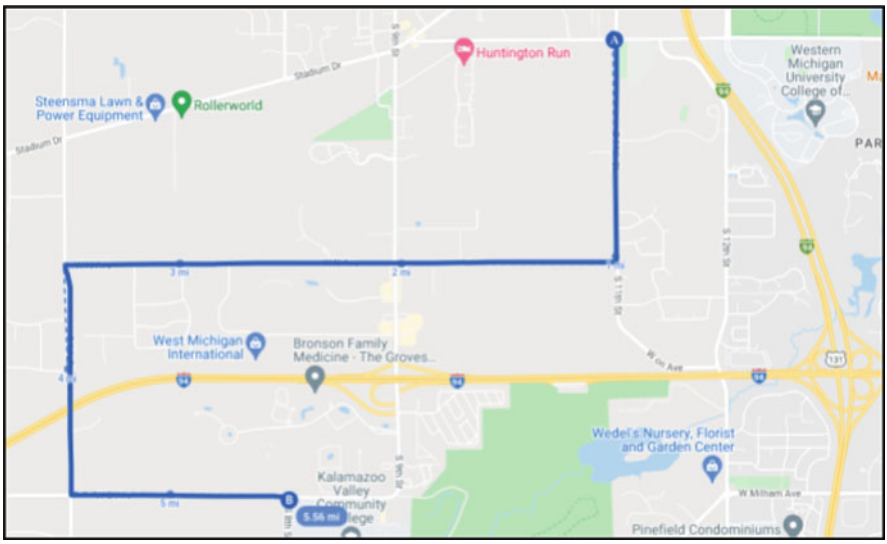


**Fig. 1** Drive cycle for data collection in Kalamazoo, MI, USA which drives from the Western Michigan University's college of engineering and applied sciences to Kalamazoo Valley Community College which totals a distance of 5.56 miles along residential roads with speed limits of 35 mph

rate than freeways and other multi-lane routes. We gathered the data during the winter of 2020. The lanes were obscured by snow and featured distinct tire track patterns, with tire tracks visible to expose the tarmac beneath . The road portion was chosen for its low traffic volume, two-lane configuration, and clearly visible lane markings during non-snowy conditions.

## *2.2 Equipment and Instrumentation*

### 2.2.1 Camera Sensor

The forward-facing ZED 2 RGB camera from Steroelabs was chosen for use in this study and is shown in Fig. 2a. The ZED 2 RGB camera was chosen firstly because it is a widely available commercial computer vision system. The ZED 2 also features a 120-degree wide-angle lens for collecting images and videos. These camera parameters are beneficial as we have a lot of information to work with, and the wide angle capability of this camera allows us to have a lot of spatial information. The camera was set to capture video at 29 frames per second at a resolution of 1280 $\times$ 720 pixels. This resolution was chosen because it was a fair compromise between image quality and image size. The ZED 2 was connected to the vehicle's onboard computer, and data was collected. The dataset was created by


**Fig. 2** (**a**) The ZED 2 camera sensor [22] and (**b**) The instrumented WMU EEAV lab research vehicles platform

segmenting and extracting frames from the recorded videos of the drive cycle. The frames from the videos show the tire tracks and features on which the model must be trained.

#### 2.2.2 Vehicle Type

The Energy Efficient and Autonomous Vehicles (EEAV) research vehicle platform, shown in Fig. 2b, was used to collect data. This platform is a 2019 Kia Niro and includes a forward-facing RGB camera, Polysync Drivekit, Neousys in-vehicle computer, vehicle Controller Area Network (CAN) bus interface and a Mobileye camera.

### *2.3 Data Pipeline*

#### 2.3.1 Data Preparation

Nearly 15,000 RGB images were acquired; however, when the images were resampled from 30 to 5 Hz, the quantity was reduced. Resampling is carried out to reduce the amount of frames for labeling, which is followed by more quality control assessments (i.e., eliminating over-exposed, occluded, or poor resolution images). This resulted in a final dataset of 1500 frames. The images were separated into three batches, each with 500 images. This was done to make the next step easier, as splitting the images into batches and obtaining labels for each batch will allow for easier error correction during the labeling process.


**Fig. 3** A raw image annotated with CVAT's interface and the corresponding ground truth label, CVAT offers multiple options such as polygons, poly lines and points to create the labeled masks

### 2.3.2 Image Ground Truth Labeling

The frames were then labeled in batches. The tire tracks on each frame were manually annotated using the Computer Vision Annotation Tool (CVAT), an open-source web tool. Each batch's labeled dataset was exported with their matching raw images in the CVAT for images 1.1 format. The raw images and an Extensive Markup Language (XML) file including the attributes for the labels, such as the position of the tire-track with their corresponding pixel location on the image, image file name, and assigned tags, were included in each exported dataset (tire-track, road, road-edge boundary). The exported labels were then used for post-processing and inputs to model training. Figure 3 shows a camera image with a CVAT toolbar and its corresponding ground truth label after CVAT annotation.

### 2.3.3 Data Conditioning

To build ML models, we must first preprocess the data and then extract features. Feature extraction is the process of transforming raw data into numerical features that the model can process while retaining original data information. This is done because it generates better results than applying machine learning straight to the raw dataset [23, 24]. Deep Neural Networks can carry out some basic feature engineering on their own as it is hard-coded into their architecture so in some cases they do not require any feature engineering at all [25].

To improve feature detection and reduce computational load, images were masked with a Region of Interest (ROI) that only included the road surface. As described in [17, 18], this is a reasonable approach because there are many methods that can detect road surface regions with high precision. We built similar road

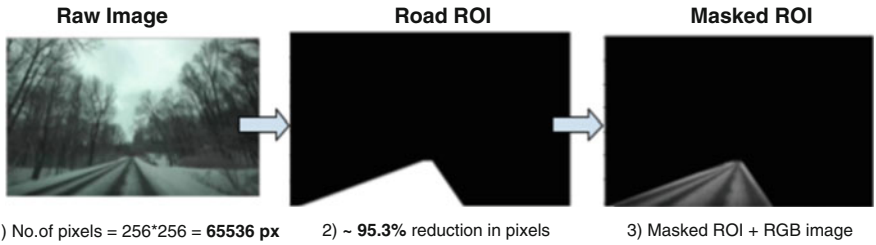


**Fig. 4** Creating the static ROI and masking the ROI onto the raw image. The raw image includes 65,536 pixels after resizing to 256 × 256, creating a static ROI which only focuses on the road removes 95.3% pixels. Finally, the raw image is masked with the road ROI to give the masked ROI

surface detection methods using a static ROI which works well for our chosen drive cycle. Figure 4 shows how to extract the ROI masked pictures.

The Road ROI is 3099 pixels in size, accounting for less than 5% of the total pixels in the raw image. Following that, the ROI mask was fused with the raw image to acquire all of the pixels contained within the ROI. This will serve as the model's input. Similar to our previous study, the different features recovered from the masked images include the red, green, blue, grayscale, and pixel X, Y values [21]. Figure 5 shows the overall process for data preparation for ML model training.

The feature vectors in Table 1 are organized into sets and selected as final inputs to the model. The results will indicate which features contribute the most to the model and perform the best. The dataset was split 55%–45% for training and testing. Input array $X = ((m \times p), n)$ was used to train the complete model where $m$ is the number of images, $p$ is the number of pixels in each image's ROI (3099 pixels for 256 × 256 images), and $n$ is the number of feature vectors in the array.

## *2.4 Classical Machine Learning Models*

### 2.4.1 Model Description

We used 6 different machine learning techniques to train the models. The first technique used is Decision Trees or **Dtrees**, which is a type of supervised machine learning technique that makes decisions and splits the dataset until all points/sets are isolated using a set of rules. The data is structured in a tree-like manner, with each dividing node representing a decision. When Dtrees is applied to our problem, it applies the rules and makes decisions based on these rules to classify pixels to be tire tracks or not tire tracks. The second technique used was Random Forest. Random forest is nothing but a number of decision trees on various subsets of the same dataset. It takes into consideration the average to improve the prediction accuracy of the dataset. The third technique used was the K-Nearest Neighbors (KNN). KNN, is based on the assumption that similar data points/classes occur

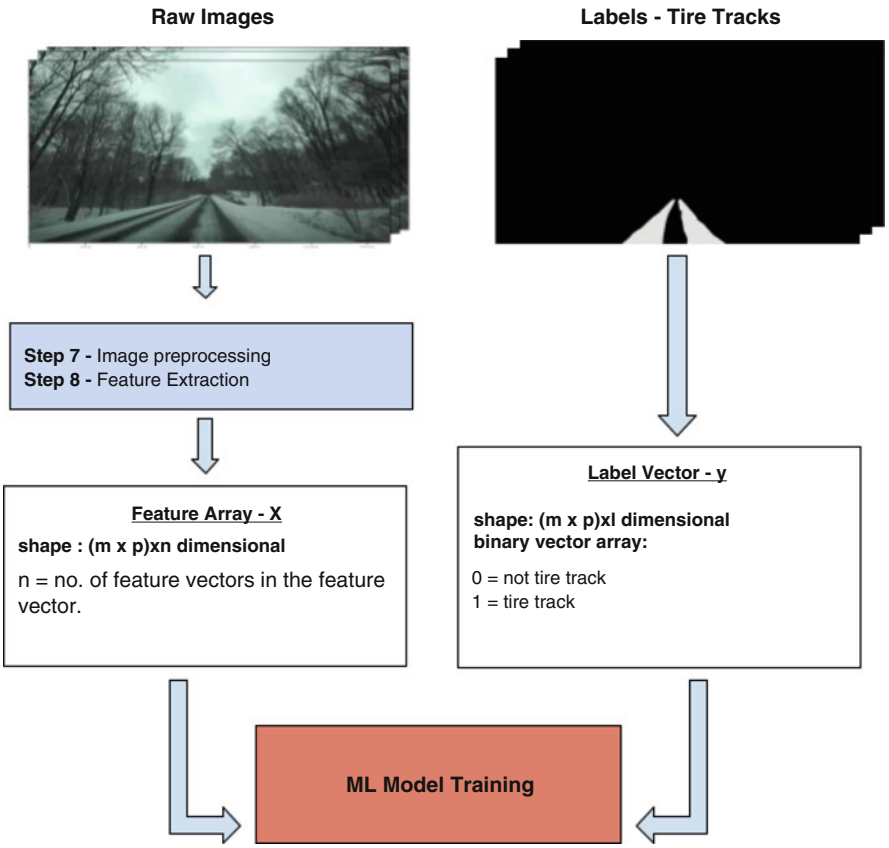


**Fig. 5** This figure summarizes the image preprocessing and feature extraction from raw images . The feature array X, which contains the raw images as well as the number of feature vectors, and the label vector y, which contains a binary array with each value representing either a tire track or not a tire track, are the two inputs to the model training. This is known as the data preparation pipeline, and it will be used in the model training section

**Table 1** Feature set properties

| Feature set | Included feature vector | Train array shape (m = 1200) | Test array shape (m = 300) |
|---|---|---|---|
| 0 | Gray | (3,718,800, 1) | (929,700, 1) |
| 1 | Gray X loc, Y loc | (3,718,800, 1) | (929,700, 1) |
| 2 | Red, Green, Blue | (3,718,800, 3) | (929,700, 3) |
| 3 | Red, Green, Blue, X loc, Y loc | (3,718,800, 5) | (929,700, 5) |

in close proximity. Classes with comparable properties are close to one another, which is the assumption by KNN. The user specifies the K value, where K is the desired number of nearest neighbors. We also used other techniques such as linear regression classifier, logistic regression classifier and naive bayes classifier. Both logistic regression and naive bayes are probabilistic classifiers, which means they calculate probabilities of each element in the dataset whereas linear regression predicts continuous values for the elements. These models were chosen for their characteristics and capabilities in commuting binary classification [26–28]. Other models such as support vector machines do not perform well with large datasets so they were not included.

### 2.4.2 Model Training

We trained a variety of machine learning models by using our input features which were defined in the data pipeline section and their associated labels. The image preprocessing and feature extraction block extracted the input feature array X and label vector y, which were then used as inputs to the machine learning model. Six distinct models, discussed in the classical ML model section were tested with each feature set (refer to Sect. 2.3.3) in order to discover the feature set/model combination that resulted in the best performance metrics.

In total we have 24 different classical ML models that can be tested. The models were trained on a desktop machine with 16 GB of RAM, an Intel i7 processor, and an Nvidia GeForce GTX 1060 graphics on Ubuntu 20.04 LTS as the operating system.

## *2.5 Deep Neural Network Models*

A wide range of tasks, including image recognition, natural language processing, and speech recognition, have been proven to be significantly improved by deep learning approaches. When compared to classical machine learning methods, deep networks scale effectively with data, do not necessitate feature engineering, are adaptable and transferable, and perform better on larger datasets with unbalanced classes [29].

CNNs are a sort of deep neural network whose architecture is designed to do feature extraction automatically, obviating the need for this step [30]. CNNs produce feature maps by performing convolutions to the input layers, which are subsequently passed to the next layer. CNNs, unlike classical machine learning approaches, can extract relevant features from raw data, removing the need for manual image processing [31, 32]. As previously indicated, our ML models were not an end-to-end pipeline for tire track detection as they required feature engineering. In this study we look at using CNN's to simplify the process and enhance overall accuracy.

Figure 6 shows a basic convolutional neural network architecture with one convolutional layer and one max-pooling layer; we will discuss more about this

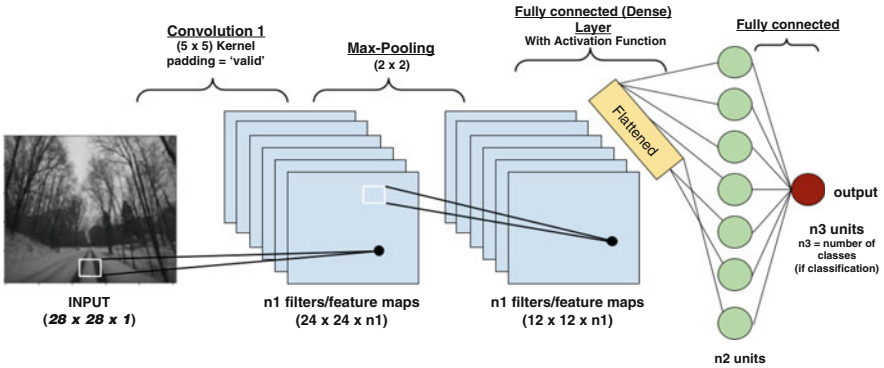


**Fig. 6** An example of a simple Convolutional Neural Network. The input image goes through a convolutional layer which has a specified kernel, the convolutional operation makes a feature map which includes important feature information from the input image. The Max-Pooling operation reduces the dimensions (halves the dimensions in this case) of the feature map. The feature maps are then flattened and passed through a fully connected layer with the output neurons equalling the number of classes/desired outputs

in the coming sections. We only focus on CNNs in context of the images to keep the discussion simpler.

Before we examine the various CNN architectures, we should examine the various types of model blocks; to simplify things, we will examine model blocks that can be combined to form various models. The convolutional block consists of a convolutional layer and a pooling layer to perform feature extraction. The convolution operation with a given filter size or a kernel size slides over the input data to perform an element-wise multiplication which is essentially matrix multiplication over the 2-dimensional data, the results inside the kernel are summed up into a single output. The pooling layer down-samples the dimensions of the feature maps, which are the outputs from the convolutional layers. The fully connected block performs classification tasks based on input from previous operations [33]. Recurrent, residual, and attention operations, explained in the next section will be added to the convolutional block to make different model architectures.

## *2.6 Model Architectures*

We have examined the fundamentals of a deep neural network in the context of images, which in our case is a convolutional neural network (CNN), as well as the numerous operations that a CNN is capable of performing. In the following subsections, a standard U-Net architecture, different convolutional model blocks such as Recurrent, Residual, and Attention, and the concept of Backbones will be discussed.

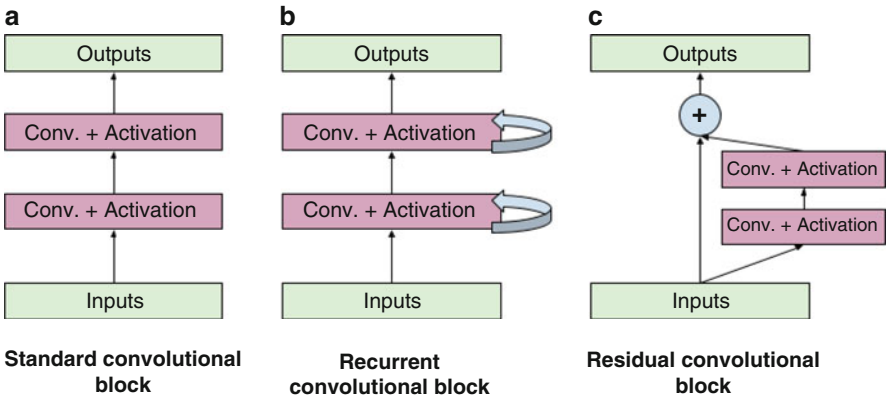


**Fig. 7** (**a**) Standard convolutional block, (**b**) Recurrent convolutional block, and (**c**) Residual convolutional block

1. Standard U-Net
2. Recurrent U-Net (Rec U-Net)
3. Attention U-Net (Att U-Net)
4. Residual Attention U-Net (Res-Att U-Net)
5. Backbone U-Net

### 2.6.1 Standard U-Net

Figure 7a shows a standard convolutional block. The two red blocks are the convolutional layers with the respective activation function such as 'ReLu' or 'Sigmoid'. The inputs to these layers are tensors of shape $(w \times h \times c)$ where $w = width\ of\ the\ image, h = height\ of\ the\ image\ , c = number\ of\ channels$ . The convolutional layers learn local patterns, which are patterns observed in the input windows. These windows are also known as kernels, and the patterns learned by these convolutions are transitionally invariant, which means that if the convolution learns one pattern somewhere, it may apply that knowledge in another place. This is why convolution layers outperform dense layers at recognizing image features. Figure 8 shows a sample code for a simple convolutional operation.

Now that we have introduced the concept of a standard convolutional block, we can look at the model architecture. The standard convolutional neural network provides an output based on the number of neurons in the output layer, if we want a binary output such as 0,1 or Cat and Dog, the output layer will only have one neuron which states that the output can only be either one of the classes. In our case, to have an end-to-end solution of obtaining tire tracks as the output image from the raw image input, we have to upsample/upscale the layers to have the same shape as the input layer and preserve the spatial information at the same time. To accomplish this, we look at a U-Net architecture. The U-Net architecture

>> model.summary()

| Layer (type) | Output Shape | Param# | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 256, 256, 3) | 0 | [] |
| lambda (Lambda) | (None, 256, 256, 3) | 0 | ['input_1[0][0]'] |
| conv2d (Conv2D) | (None, 256, 256, 32) | 896 | ['lambda[0][0]'] |
| dropout (Dropout) | (None, 256, 256, 32) | 0 | ['conv2d[0][0]'] |
| conv2d_1 (Conv2D) | (None, 256, 256, 32) | 9248 | ['dropout[0][0]'] |

**Fig. 8** Standard keras model summary for a standard convolutional block in a U-Net architecture

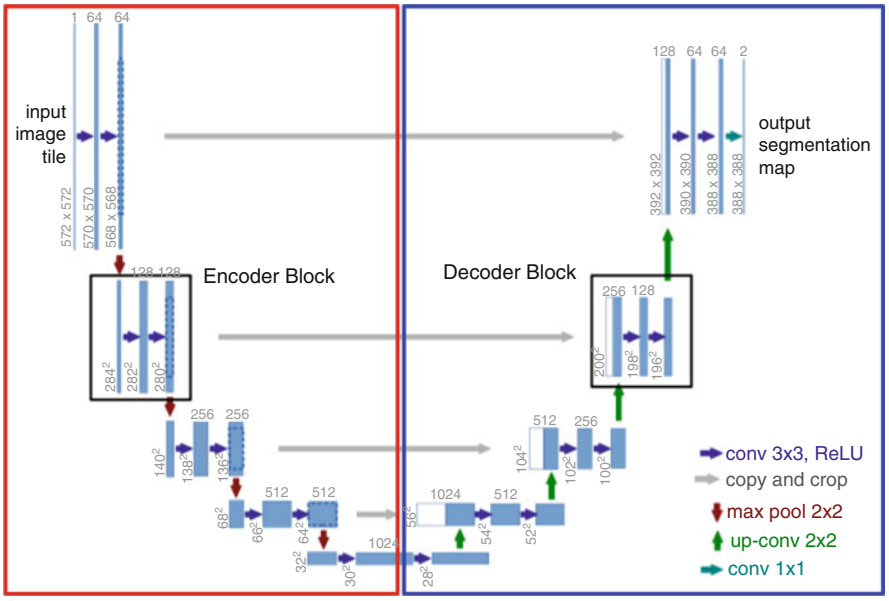


**Fig. 9** Standard U-Net architecture from [35] modified to support the discussion

has been shown to perform exceptionally well in computer vision segmentation [34]. CNN's fundamental assumption is to learn the feature mapping of an image and then utilize that knowledge to construct more sophisticated feature maps. This technique is effective for classification problems since it converts the image to a vector, which is subsequently classified. However, image segmentation requires not only the transformation of a feature map to a vector but also the reconstruction of an image from the vector. Figure 9 shows the standard U-Net architecture. The red box shows the encoder path and the blue box shows the decoder path.

A standard convolutional block can either serve as an encoder or a decoder. The encoder path makes the input array smaller (also known as downsampling) with

every max-pooling operation and doubles the feature maps. Conversely, the decoder path scales the input back to its original shape with every up-convolution operation.

While converting an image to a vector, the U-Net architecture learns the image's feature maps, which are then utilized to convert it back to an image. The contracting path or the encoder path is on the left side of the U-Net architecture, while the expansive path or the decoder path is on the right. After each downsampling block, the number of feature channels/filters doubles in order to learn more intricate structures from the previous layer's output, while the image size reduces. This path is filled with numerous contraction blocks. Each block accepts the input and applies it to a $3 \times 3$ convolutional layer (where $n \times n$ is also known as the kernel, $n$ can be any number, usually it is common to see $n = 3$ or 5) and with an activation function and padding (usually rectified linear unit or 'ReLU'). A $2 \times 2$ max-pooling layer is used for downsampling. We begin with 32 feature channels and increase them by a factor of two with each contraction block until we reach 512 feature channels, at which point we reach the expansive path. Each block in the expansive path (shown on the right) is composed of two $3 \times 3$ convolution layers and one $2 \times 2$ upsampling or up-convolution layer with an activation function and padding. The input is concatenated by appending the feature maps of the matching encoder block to the corresponding decoder block as represented by the gray arrow connecting the two layers. Each block in the expansive path reduces the number of feature channels by half. In the final layer, a $1 \times 1$ convolution layer is applied, with the number of feature maps corresponding to the number of needed classes/segments. Additionally, we add a dropout layer between each convolution layer in the encoder and decoder blocks to combat overfitting. Note the number of feature channels and input size shown in the figure are not the same for every model. Depending on the requirements such as the input shape, the kernel size, feature channels, the parameters can be modified in the architecture.

These general concepts of how a convolutional layer works and how it's used in a neural network architecture like a U-Net to achieve image segmentation are important for development of the Recurrent and Residual Deep Neural Networks discussed next. We will now discuss the various convolutional blocks and operations that will result in different model architectures.

### 2.6.2 Recurrent U-Net

Figure 7b. shows an example of a recurrent convolutional block; the recurrent network can store information over time by using the feedback connection represented by the arrows on the convolution layer. Even though the input is constant, the network in a recurrent convolutional layer can evolve over time. We can specify the number of iterations that the recurrent block must undergo. We simply substitute the standard convolution blocks with recurrent convolutional blocks in the encoder and the decoder path.

Figure 10 shows a sample code for a recurrent convolutional operation. If we combine the recurrent convolutional block with a standard U-Net we get a recurrent

>> model.summary()

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 256, 256, 3)] | 0 | [] |
| conv2d (Conv2D) | (None, 256, 256, 32) | 128 | [‘input_1[0][0]’] |
| conv2d_1 (Conv2D) | (None, 256, 256, 32) | 9248 | [‘conv2d[0][0]’] |
| dropout (Dropout) | (None, 256, 256, 32) | 0 | [‘conv2d_1[0][0]’] |
| add (Add) | (None, 256, 256, 32) | 0 | [‘dropout[0][0]’, ‘conv2d[0][0]’] |
| conv2d_2 (Conv2D) | (None, 256, 256, 32) | 9248 | [‘add[0][0]’] |
| add_1 (Add) | (None, 256, 256, 32) | 0 | [‘conv2d_2[0][0]’, ‘conv2d[0][0]’, ‘conv2d_3[0][0]’, ‘conv2d[0][0]’] |
| conv2d_3 (Conv2D) | (None, 256, 256, 32) | 9248 | [‘add_1[0][0]’] |
| max_pooling2d (MaxPooling2D) | (None, 128, 128, 32) | 0 | [‘add_1[1][0]’] |

**Fig. 10** Model summary of a recurrent convolutional operation

convolutional U-Net (RCU-Net) which is shown in Fig. 11. In Fig. 11 we can see that the recurrent convolutional layers replace the standard convolutional layers to make the RCU-Net.

The recurrent convolutional layers will look at the same features throughout the provided recurrency number, in our instance the layers will look at the same characteristics of pixels having a tire track multiple times, which will help the model reinforce when its learning process is taking place.

### 2.6.3 Attention U-Net (Att U-Net)

In image segmentation training, attention is used to highlight only relevant activations. This saves processing resources and improves the network’s generalization power. Basically, the network may “focus” on selected areas of the image. We use Soft attention. Soft attention weighs different parts of the image. High relevance areas are given to areas of higher weight, whereas low relevance areas are given a lower weight. As the model learns, higher weighted regions get more attention [36, 37].

Figure 12 shows the overall layout of an attention gate along with the gating signal ($g$) and skip connection ($x$) Two inputs are required for the attention gate: $x$ and $g$, $g$ is the gating signal that originates at the network’s sub-layer. Since g originates from a deeper layer of the network, it contains a more complete

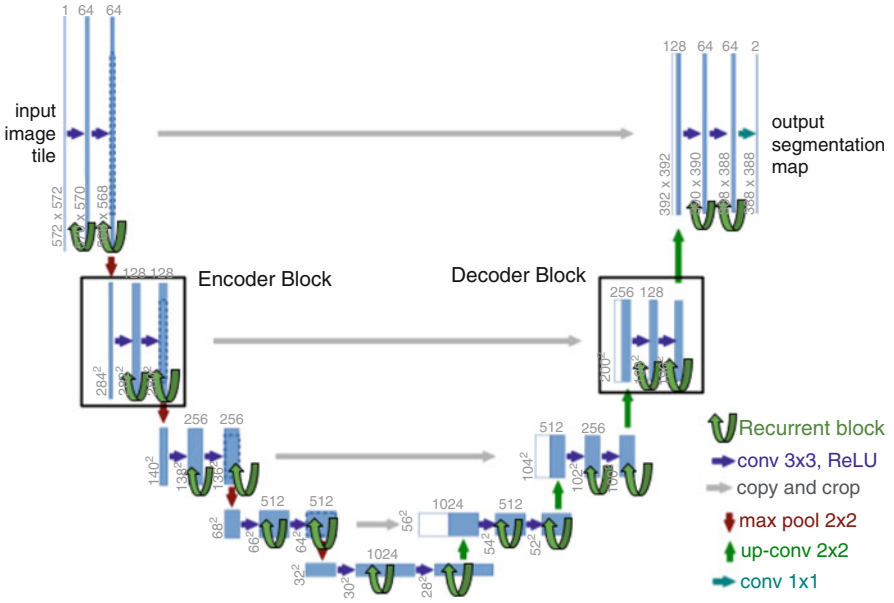


**Fig. 11** Recurrent U-Net architecture obtained from modifying the standard U-Net by replacing the standard convolutional blocks with recurrent convolutional blocks, original figure modified from [35]

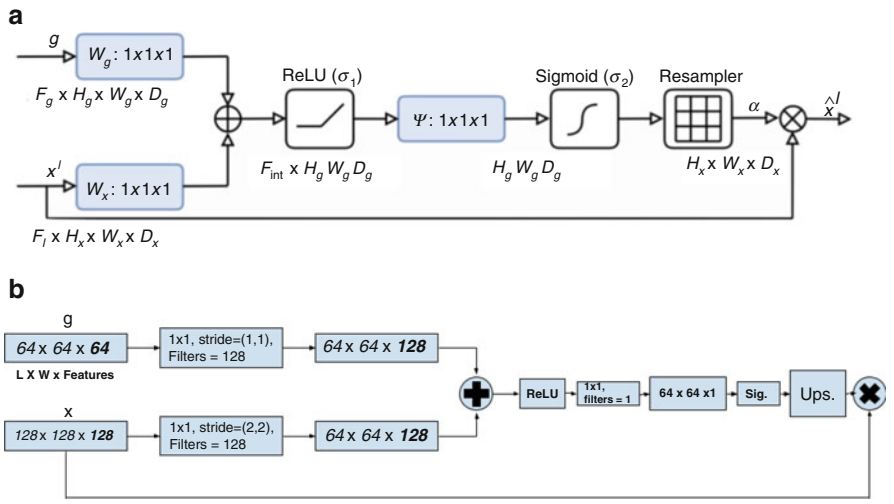


**Fig. 12** (**a**) Attention gate, obtained from [37] and (**b**) Attention gate with two inputs $x$ and $g$ having different input dimensions

representation of features.While $x$ originates in the early levels (concatenation of encoder blocks), and so contains more spatial information. Consider the first

attention gate, which is at the topmost part of the decoder block (output layer). Input $x$ is the encoder block's output, which is $64 \times 64 \times 64$ ($height \times width \times filters$). The output from the preceding layers (decoder block) is input $g$, which has dimensions of $128 \times 128 \times 128$ ($height \times width \times filters$). To make $x$ have the same dimensions and feature numbers as $g$, we pass it through a convolutional layer with a stride of (2, 2) and a filter count of 128, halving the dimensionalities while maintaining the same filter count for both $x$ and $g$. We can perform the operations on both inputs because they have the same dimensions. The addition operation adds aligned weights and makes them larger. Upsampling is used to restore the dimensions to their original values ($128 \times 128$ in this case). Finally, the output of the upsample is multiplied by the input $x$ to perform the attention operation. Figure 12 summarizes the operation performed by the attention gate.

If we combine the attention operation with a standard U-Net we get an Attention U-Net which is shown in Fig. 13. Since we are using soft attention, the key activations would be the contrasting regions between tire tracks and the snowy road surface.

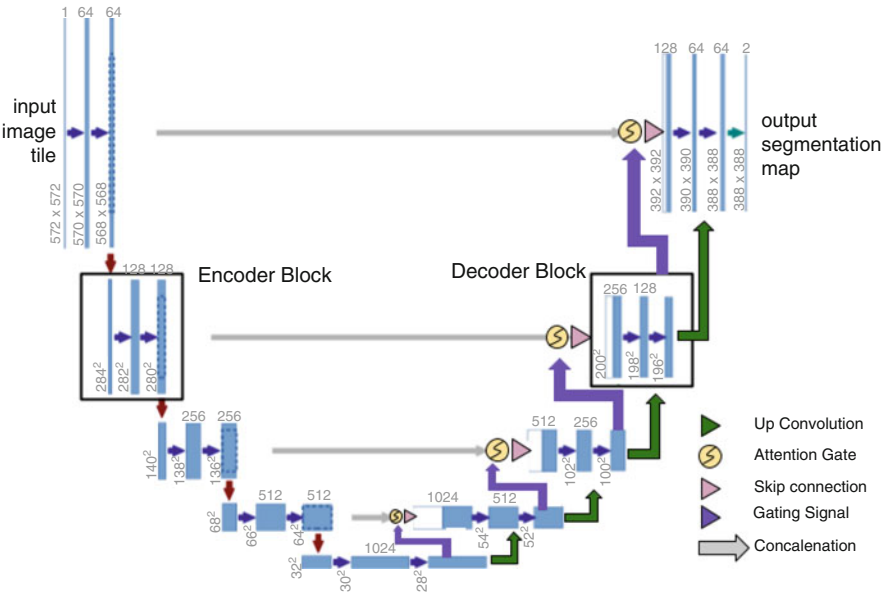


**Fig. 13** Attention U-Net architecture obtained from modifying the standard U-Net by adding attention gates and skip connections to each convolutional block in the decoder path, original figure modified from [35]

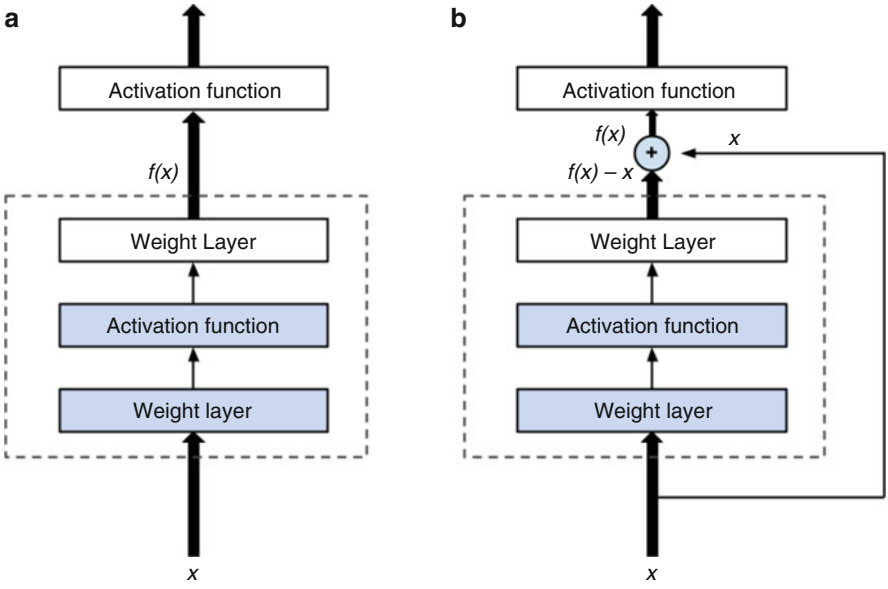


**Fig. 14** (**a**) Traditional network with a single input which goes through the weight layers and specified activation functions such as 'ReLu' and (**b**) Network with residual function which uses the idea of skip connections to learn from inputs provided by previous layers

### 2.6.4 Residual Operation

Having more convolutional layers and making the model deeper hurts the generalization ability of the network which causes overfitting. To address this issue we use the residual operation which is shown in Fig. 7c. The residual network addresses this issue by introducing the concept of skip connections [38]. The skip connections address the vanishing gradient problem. One group of researchers [39] discusses this problem and how Residual-Net reduces the risk of overfitting and smoothens the loss surfaces [39]. Figure 14a shows the traditional feedforward network, where the block is trying to learn $f(x)$, so learning true output $f(x)$, whereas the residual block in Fig. 14b is trying to learn the residual $R(x) = f(x) - x$. The x which is being added to the residual from the input is also known as the identity. So essentially, in networks with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away. Inputs can forward propagate faster through residual (shortcuts) across layers.

### 2.6.5 Residual + Attention U-Net (Res-Att U-Net)

Additionally, it is possible to combine two distinct blocks, such as a residual convolutional block with an attention operation. This generates a Residual Attention

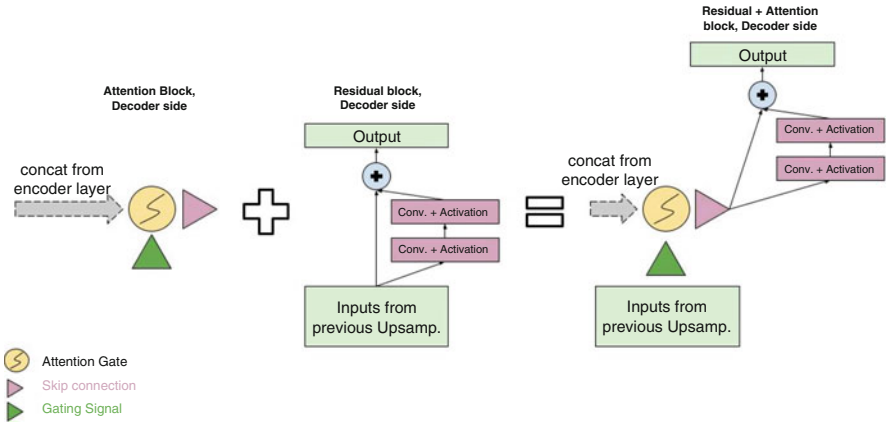


**Fig. 15** We can combine two operations such as attention and residual. This results in the Residual Attention block or Res-Att. block. This block can only be used on the decoder path as it needs the spatial information from the previously concatenated layers by the use of skip connections

Convolutional Neural Network, or ResAtt-U-Net. Figure 15 illustrates the combination of the attention block and the residual convolutional block. The residual convolutional blocks can be substituted for the standard convolutional blocks on both the encoder and decoder ends of the model, whereas the attention operation can only be applied to the decoder path/blocks. And hence, the encoder path contains the residual convolutional blocks and the decoder path contains the Residual + Attention convolutional blocks.

Figure 16 shows the architecture for the ResAtt U-Net. Combining attention gates with residual convolutional blocks could increase the model's ability to detect features and reduce overfitting. This should improve the model's ability to generalize image feature recognition, in our instance tire track detection, with little overfitting.

### 2.6.6 Backbone U-Net

Another way of making model architectures is by using backbones. Backbones are pre-made architectures that can be used to replace the encoder path of our U-Net. A few of them are VGG, ResNet, and Inception [40]. These backbones are trained on datasets for example ImageNet [41] and we can benefit from transfer learning by using the pre-trained weights.

We used the segmentation models library that contains various Python libraries with Neural Networks for Image segmentation tasks[40]. This library consists of 4 model architectures for binary and multi-class image segmentation. Each architecture has 25 available backbones. All backbones have pre-trained weights for faster and better convergence. We used the resnet34 as our model architecture

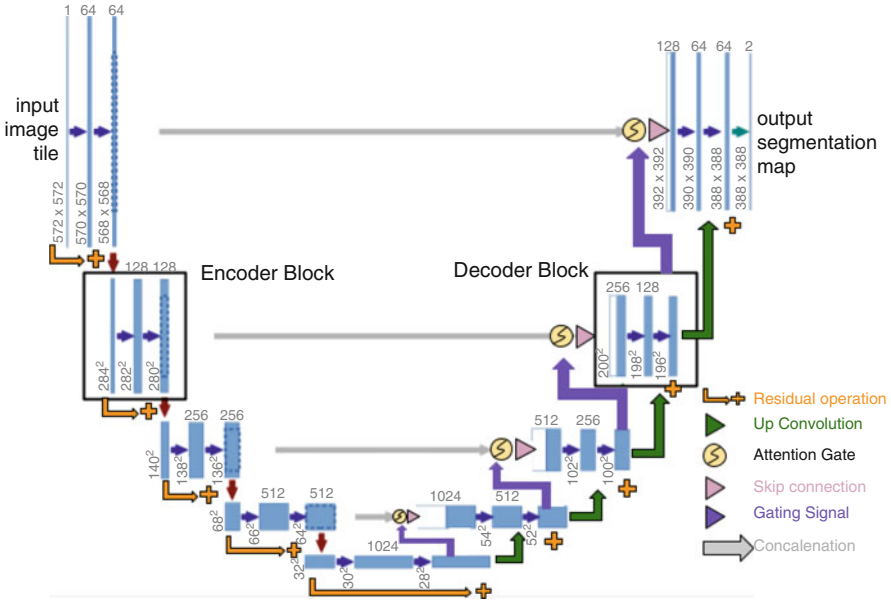


**Fig. 16** Residual attention U-Net or Res-Att U-Net architecture obtained from adding residual blocks to the encoder path and Res-Att. blocks to the decoder path mentioned in Fig. 15, original figure modified from [35]

```
>> BACKBONE = 'resnet34'
>> model = sm.Unet(BACKBONE,
      classes=1,
      activation='sigmoid',
      encoder_weights ='imagenet')
```

**Listing 1** Model backbone and encoder weights used from segmentation models library

and ImageNet as encoder weight. ResNet34 is a 34-layer residual network [38, 41]. ImageNet is a large dataset containing over 1000 classes, 1.28 million training images, and 50 thousand validation images. The encoder weights which are set to ImageNet are the pre-trained weights from the same network, which will make training faster. Listing 1 shows the model backbone and encoder weights used from the segmentation models library.

## *2.7 Model Training*

The inputs to the model are an image with (width x height x channels). As we are using the raw RGB image (feature set 2, refer to data pipeline section) which has been resized to the desired size for training. In our case, the inputs are of shape (256

```
>> import tensorflow as tf
  from tensorflow import keras

>> model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
>> model.compile(optimizer = 'adam',
loss = 'binary_crossentropy',
metrics = [IoU, tf.keras.metrics.Accuracy(),
      tf.keras.metrics.Recall(),
      tf.keras.metrics.Precision()])
```

**Listing 2** Lines of code used for compilation of CNN models

$\times$ 256 $\times$ 3). Unlike the classical machine learning models, no feature engineering is used to train the CNN models, we can directly feed in the raw RGB image as the input to the model. We resize the images to make the training process faster and is a standard practice while training CNNs. We split the dataset into 1200 images for training and 300 images for testing. We compiled the 5 CNN models with the same optimizer, loss function and metrics. We set the optimizer to 'adam' and the loss function as 'binary cross entropy', both have been applied successfully to similar semantic segmentation tasks [42–44]. Listing 2 shows the line to compile the CNN models.

We can evaluate both the classical ML models and the different CNN models using different metrics. These metrics should serve as good evaluations to test the output of the predicted model $y_{pred}$ with the ground truth. Intersection over union (IoU), pixel prediction accuracy, precision, recall, F1 score, and frame per second (FPS) were the evaluation metrics. These measures were evaluated based on the ability to make conclusive inferences from the performance of the model [26]. Below are the equations explaining these metrics and the four corners of a confusion matrix, which determine the true positives, true negatives, false positives, and false negatives, respectively. We only predict tire tracks, hence it's a binary classification task, hence classes = 1

1. True Positive (TP): no. of pixels which were a tire track and correctly identified as a tire track
2. False Positive (FP): no. of pixels which were not a tire track but identified as a tire track
3. True Negative (TN): no. of pixels which were not a tire tracks and identified as not a tire track
4. False Negative (FN): no. of pixels which were a tire track but identified as not a tire track

$$\text{Accuracy} = \frac{\text{total correct predictions}}{\text{all predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$IoU(Intersection\ over\ Union) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2}$$

$$mIoU = \frac{1}{n} \times \sum_{n=1}^{n} \frac{intersection}{union} = \frac{1}{n} \times \sum_{n=1}^{n} \times \frac{TP_i}{TP_i + FP_i + FN_i} \tag{3}$$

where $n$ is the number of classes

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F/F1\ Score = 2 \times \frac{precision \times recall}{precision + recall} \tag{6}$$

The accuracy Eq. 1 is the proportion of total accurate predictions made by our model over all the predictions. But accuracy alone does not tell the whole story when working with a dataset with an imbalance class distribution [45]. Accuracy is calculated over all classes. In our sample, there is a significant imbalance between the tire tracks and not tire tracks (background), therefore accuracy is not an appropriate evaluation metric. In terms of pixel-wise accuracy, this implies that the inaccuracy of minority classes is dominated by the accuracy of majority classes. IoU, also known as the Jaccard Index or the Jaccard coefficient, is significantly more indicative of success for segmentation tasks, particularly when input data is sparse and there is a high class imbalance. When training labels consist of 80 to 90% background and a small number of positive labels, a basic metric such as accuracy can acquire a high score by being dominated by the larger class. This naive problem will never arise with IoU, since IoU is unconcerned about true negatives, even with incredibly limited data. IoU calculates the overlapping region for the true and predicted labels by comparing the similarity of finite sample sets $A$, $B$ as the IoU [46]. According to Eq. 7, $T$ represents the true label image and $P$ represents the output prediction. This is used as a measure, giving us a more precise means of quantifying IoU in the segmentation region of our model. The mIoU or mean intersection over union is nothing but the IoU computed over each class. We would only be looking at IoU because we only have one class.

$$Jaccard\ Index\ (IoU) = \frac{|T \cap P|\ (Area\ of\ Overlap)}{|T \cup P|\ (Area\ of\ Union)} \tag{7}$$

Listing 3 shows the implementation of IoU as a metric in the model and then used to compile the model.

## 2.8 Results

In this section, we will set forth the results, beginning with the metrics for the different ML models and their feature sets, and then moving on to the metrics for

```
>> from tensorflow import keras
>> def IoU(y_true,y_pred):
        y_true_f = keras.backend.flatten(y_true)
        Y_pred_f = keras.backend.flatten(y_pred)
        inter = keras.backend.sum(y_true_f * y_pred_f)
        return (inter + 1.0)/(keras.backend.sum(y_true_f) +
    keras.backend.sum(y_pred_f) - inter + 1.0)
```

**Listing 3** Jaccard coefficient/ Intersection over Union (IoU) as a metric

the CNN models. As described in the previous section, IoU is the relevant metric since, unlike accuracy, it provides better and complete information about the model.

### 2.8.1 Classical Machine Learning Models

We obtained the metrics for the 24 different model combinations, which included the 6 different ML models with 4 feature sets each. We are mainly interested in IoU scores for each model. We used the standard scaling method to plot the IoU of each model and feature set as shown in Fig. 17, where $Standard\ scale\ value = (IoU_x - IoU_{mean})/IoU_{std.dev}$. The random forest model performed the best using feature set 1 containing grayscale pixel values and pixel X,Y locations as the feature set input. All models that use pixel locations outperform those that do not. In

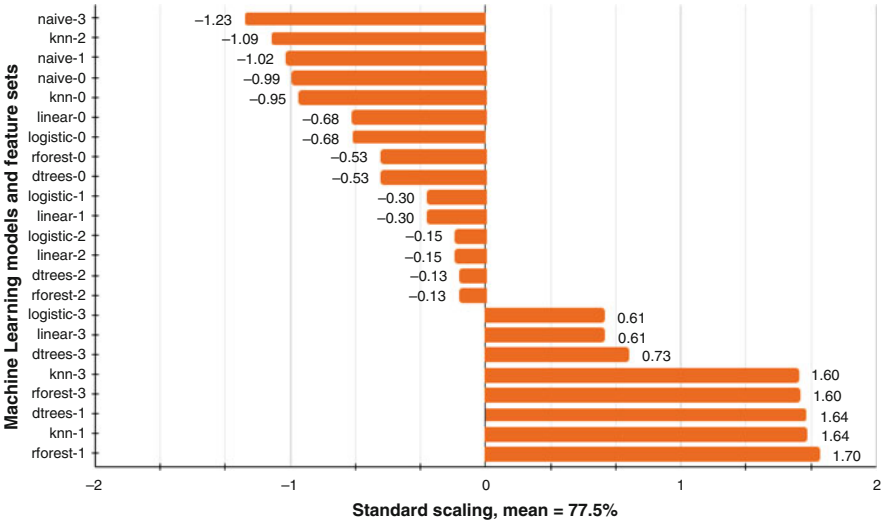


**Fig. 17** Standard scaled IoU for all the classical ML models, standard scaling centers all the values around the mean with a unit standard deviation. The model/feature set combinations with positive values are good performing models, where Random forest with feature set 1 obtains the highest IoU score. This technique allows us to rule out models that perform poorly

addition, the image demonstrates that grayscale pixels provide a higher IoU than RGB pixels, as the three highest-performing models are all grayscale. Random forest seems to be the most effective method for every feature set. This is another indicator that feature engineering improves the performance of our machine learning models.

#### 2.8.1.1 Performance Comparison Between Classical ML Models

Figure 18 shows the metrics for the best performing classical ML models. KNN with feature set 1 obtained an IoU score of 83.2%, Accuracy of 90% and an F1 score of 91.0%. Naive Bayes with feature set 0 obtained an IoU score of 74.1%, Accuracy of 82% and an F1 score of 85.1%. Random Forest with feature set 1 attained the highest IoU score at 83.4% with an Accuracy of 90% and F1 score of 91%. From an initial analysis this might indicate that Random forest with feature set 1 is the best performing model/feature-set combination. Decision trees with feature set 1 follows Random forest with an IoU score of 83.2%. Regression based classifiers such as linear regression classifier and logistic regression classifier achieved the same scores and performed well on feature set 3. Both of these models needed more feature information than the other models.

Random Forest with feature set 1 performed best in terms of key metrics like IoU, Accuracy, and F1 score, followed by Decision trees with feature set 1. As described in section 2.8, the IoU score provides a more comprehensive assessment

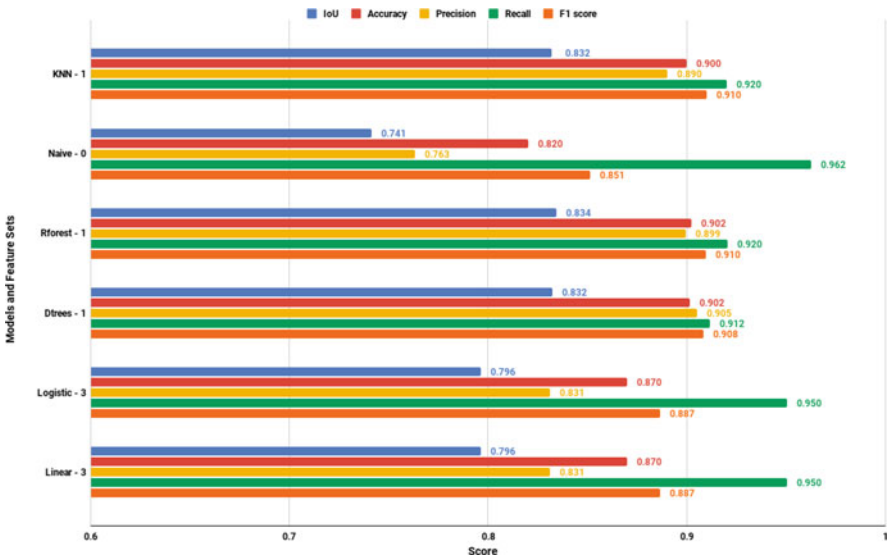


**Fig. 18** Classical ML model metrics for the best performing models, where models with high IoU score, Accuracy and F1 score are of interest

of a model's performance. A high training IoU score indicates that there is a greater overlap between the predicted and ground-truth tire track pixels. Since accuracy is calculated across all classes, it does not account for the imbalance between classes and is not the metric of interest. By computing their harmonic mean, the F1 score accounts for both precision and recall. When other metrics are taken into account, random forest, decision trees, and KNN achieve a high F1 score.

#### 2.8.1.2 Real Time Compute Speed Comparison

We may state that models like Random forest, Decision trees, and KNN, along with their provided feature sets, are suitable for our application based on the previous metrics, however real-time computation is important as well since the inability to provide outputs in time removes the approach from realistic implementation. In our case, we can use the relationship between compute speeds and feature sets to determine the best model/feature-set combination. The model with the greatest IoU score performed poorly in real-time computation at 11.3 FPS, whereas Decision Trees, which achieved an IoU of 83.2%, just 0.2% below the best model, performed at 1084 FPS. KNN, which performed well on key metrics, struggled in real-time compute performance. Based on the metrics and real-time compute speed, we can say that Decision trees with feature set 1 is a good fit for our application. The real time compute speeds for all the models is shown in Fig. 19.

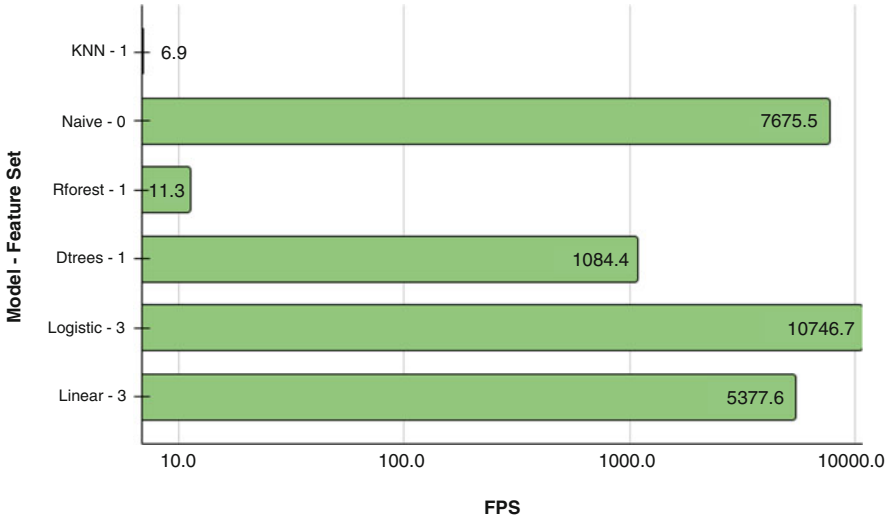


**Fig. 19** Real time compute speeds in FPS for the best model/feature set combination. Low computational cost algorithms have a high FPS and high computational cost algorithms have a low FPS. More efficient models might yield faster a FPS score

**Fig. 20** Qualitative prediction from our classical ML model. This was produced by overlaying the predictions from the Decision trees with feature set 1 onto the raw image

In addition to quantitative analysis, we must also consider qualitative analysis for the models. Performing both of these procedures will ensure a thorough review of the models and aid in selecting the most appropriate model for our application. Figure 20 displays a qualitative model output. The anticipated array of tire track pixels within the ROI was then overlaid on the raw image. This was derived from Decision trees with feature set 1, our most effective ML model.

### 2.8.2 Convolutional Neural Network Models

The CNN model's output is shown in Fig. 19; all of the models will produce an image that reflects the segmentation mask for the predicted tire track. Unlike the classical ML models, where the output is a flattened array of points which include the prediction values for each pixel in the ROI, the CNN models output a segmentation mask of the predicted tire track. Semantic segmentation means that each pixel is assigned a label based on the prediction. The output from the CNN models gives out a segmentation mask which is of the same image as the input to the model which tells us where the tire tracks lie given a new image. These prediction masks can be used to obtain pixel values in terms of labels for the image. By changing the input dimensions of the image, we can obtain a predicted segmentation mask with the same input dimensions. Figure 21a shows the raw image which is the input to the model obtained from the test set, this image was resized to the shape of $256 \times 256$ to make the prediction faster. Figure 21b shows the ground truth label

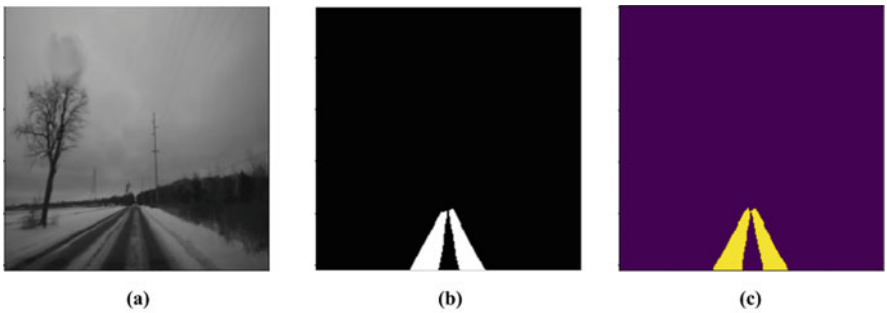


**Fig. 21** (**a**) Raw image, (**b**) Ground truth label and (**c**) Predicted tire track. The CNN prediction is the image of the segmentation mask with the same size as that of the input image

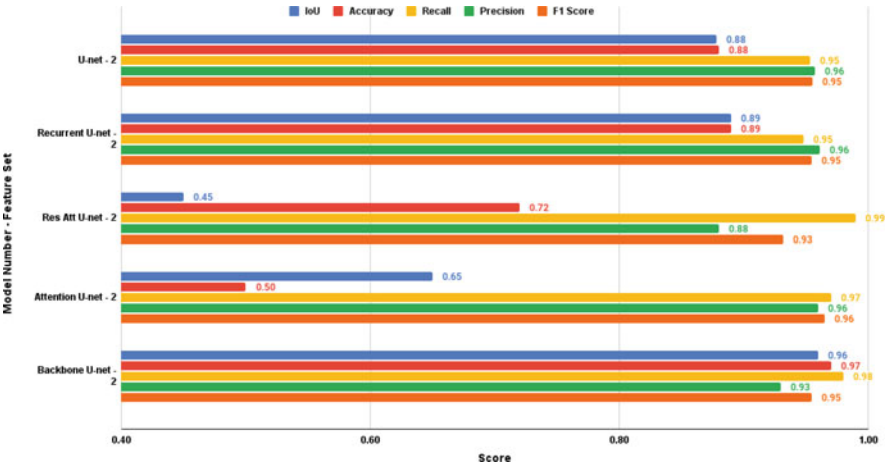


**Fig. 22** CNN model metrics for the best performing models, where models with high IoU score, Accuracy and F1 score are of interest

that was annotated using CVAT and Figure 21c shows the output from the standard CNN U-Net model. The prediction resembles the ground truth label.

#### 2.8.2.1 Performance Comparison Between CNN Models

The metrics for each CNN model are displayed in Fig. 22. All of the CNN models use feature set, as mentioned previously, CNN models do not require feature engineering, the input to the models is the raw image, which is feature set 2. The Standard U-Net model obtains an IoU score of 88%, Accuracy of 89%, and F1 score of 95%. The Recurrent U-Net model achieved an IoU score of 89%, Accuracy of 89% and F1 score of 95%. The Residual Attention U-Net and the Attention U-Net both performed poorly in terms of IoU and Accuracy. The Backbone U-Net

attained the highest IoU, Accuracy and F1 score among all the other models. This might indicate that the Backbone U-Net model is the best performing CNN model. However, real time compute speeds also need to be considered as part of a qualitative analysis.

#### 2.8.2.2 Real Time Compute Speed Comparison

Figure 23 shows the real time compute speed of the five different CNN models. The Recurrent U-Net model achieved the fastest real-time compute speeds, followed by the U-Net. Backbone U-Net, which had the best IoU score, had the slowest compute speed of 25 FPS. A qualitative investigation is required to determine which model produces good results.

The outputs from all of the CNN models on new images are shown in Fig. 24, along with the IoU score earned on each of the models during training. On the training set, all of the models perform well, but when tested on new images, the results in Fig. 24 demonstrate which model produces good results. Model 1 and 2 perform well and output diverse tire tracks as their predictions complement their IoU scores. Models 3 and 4 have poor performance. Model 5, which has the highest IoU, performs well, but it has a tendency to overfit the tire tracks by merging the space between them and does not distinguish between the left and right tracks like models 1 and 2. This could also explain why Model 5 has the highest IoU score and shows evidence of overfitting. Looking at the real time compute speeds, both Model 1 and 2 perform better then model 5. Based on the metrics and real time compute

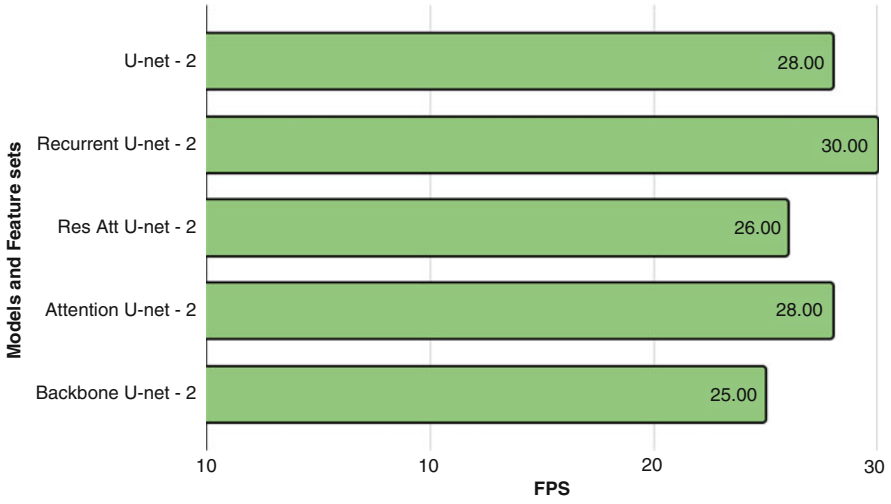


**Fig. 23** Real time compute speeds in FPS for the best CNN model. Low computational cost CNN models have a high FPS and high computational cost algorithms have a low FPS. More efficient architectures might yield faster a FPS score

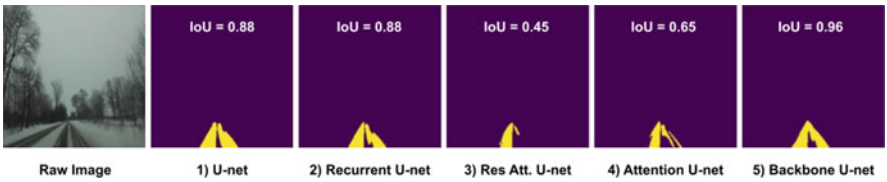


**Fig. 24** Qualitative analysis of the outputs from the 5 CNN models. A high IoU score means the model performs better, which is true in case of models 1 and 2, their outputs show distinct tire tracks. The highest IoU which is attained by model 5 shows signs of over fitting as the left and the right tracks have merged into one solid body. Models 3 and 4 with low IoU scores show poor tire tracks

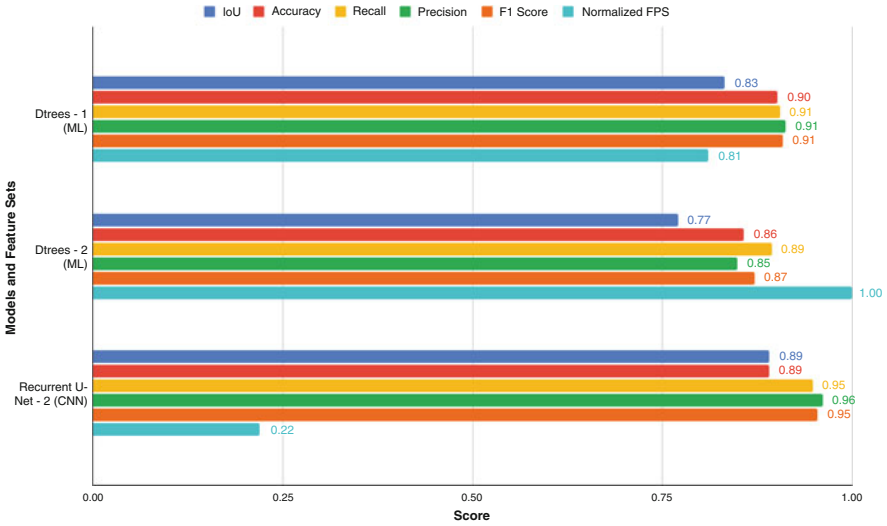


**Fig. 25** Best classical ML model and best CNN model metrics comparison. The FPS values have been normalized between 0 and 1. The CNN model performs much better in terms of IoU, Accuracy and F1 score without using any kind of feature engineering. The classical ML models outperform the CNN model in real time compute speeds (FPS)

speeds shown in Fig. 22 and Fig. 23, and a qualitative analysis shown in Fig. 24, we can conclude that Recurrent U-Net is a good fit for our application.

### 2.8.3 Best ML Models vs Best CNN Model

When comparing the best model from the classical ML model section, Decision Trees with feature set 1, to CNN models that use feature set 2, we should also compare Decision Trees with feature set 2, which is the raw RGB image as input. We compare these to the Recurrent U-Net, which is the best performing CNN model. We look at all the key metrics and normalized real time compute values.

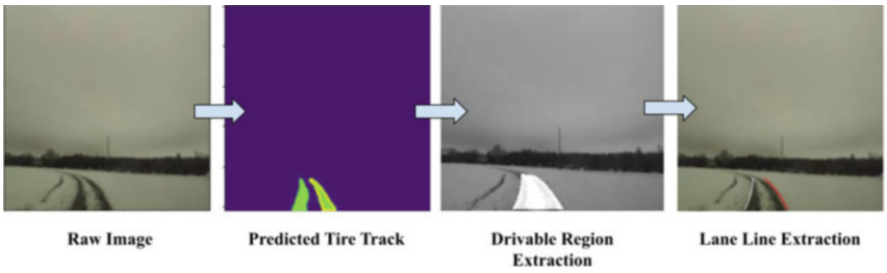


**Fig. 26** The overall process of using this system to obtain the drivable region. By implementing a few CV transformations, we can extract the drivable region from tire tracks, this can be further expanded to get lane line information

Figure 25 shows that the CNN performs much better at metrics such as IoU, Accuracy, Recall, Precision, and F1 score. To perform a fairer comparison, Decision Trees with feature set 2 and the Recurrent U-Net with feature set 2 should be compared, as both have the same feature sets. Recurrent U-Net outperforms the Decision Trees in all of the key metrics except for real time compute speeds.

## *2.9 Drivable Region Extraction from Tire Tracks*

Once the tire tracks are identified, the drivable region can be extracted using standard computer vision transformations. Figure 26 illustrates an example of overlaying the predicted tire tracks on the raw image to generate the drivable region. Likewise, we can extract the lane lines. Our results show that using tire tracks, we have an alternate method in obtaining the drivable region unlike the predictions from the leading CV provider.

Figure 27 depicts the three cases: (a) Detections from the leading CV provider without lane line occlusion. (b) Detections from the leading CV provider with snow occlusion on lane lines and (c) Detections from our algorithm to extract the drivable lane (Fig. 26). In Fig. 27a, the leading CV provider is able to detect the lane lines, which are indicated by the two green lines that show the left lane line and right lane line while the third red line indicates the road boundary. In Fig. 27b both the left and right lane lines appear red, indicating that the system lacks confidence in detecting the lane lines. Figure 27c shows the drivable lane detection from our model.

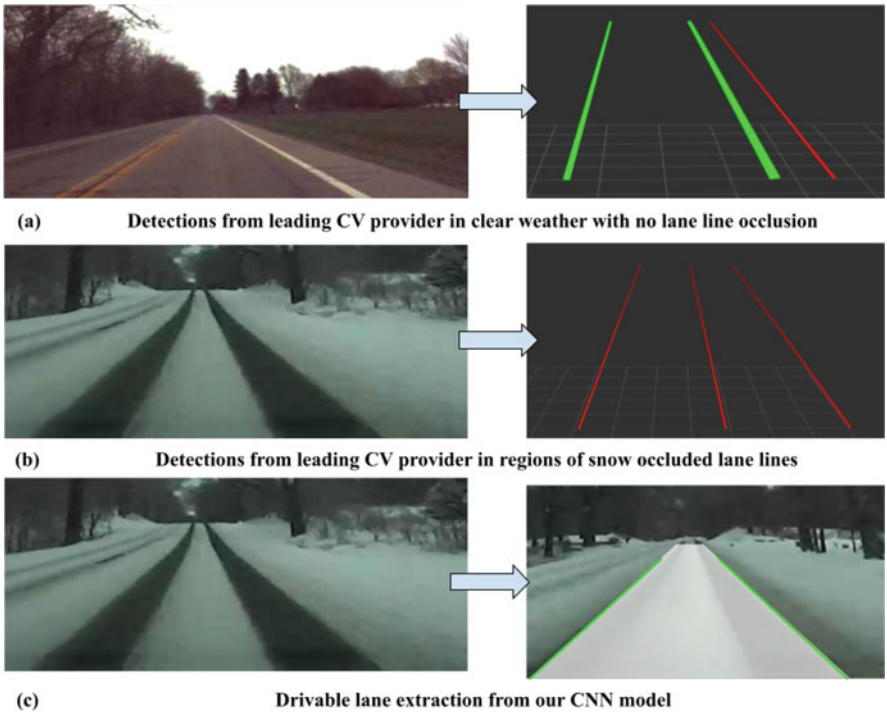


**Fig. 27** We looked at a road section from our drive cycle, where we collected camera data and detections from a leading CV provider in two conditions (1) clear weather with no lane line occlusion and (2) snowy weather with lane line occlusion. In (**a**) we can see that the leading CV provider system is able to detect lane lines with full confidence. In (**b**) the system is misidentifying lane lines and has poor confidence in detecting the drivable region whereas in (**c**) our algorithm is able to detect the drivable region using the predictions and transformations

## 3 Conclusion

This study investigates the research gap in driveable region detection for snow-covered roads with a single camera sensor that can be incorporated in current ADAS systems. We proposed a new method for identifying the drivable region in snowy road conditions when lane lines are occluded by focusing on tire tracks and extracting the drivable region with that information. Data was first acquired using our instrumented vehicle, and then processed by extracting frames from videos, segmenting them into batches, and labeling them with CVAT. That data was then utilized to build a CV model. We explored both classical ML approaches and Deep Neural Networks, specifically CNN, for detecting the driveable region based on tire tracks. We developed 5 different neural network architectures and compared their performance to that of classical machine learning methods. We evaluated the U-Net based CNN models for IoU, Accuracy, Recall, F1 score, and FPS using only the

raw image with no image pre-processing or feature extraction. The Recurrent U-Net model had an IoU score of 89%, followed by the U-Net model which achieved 88%. The best performing ML model was Random Forest with feature set 1 with an IoU of 83.4%, however when we looked at the FPS, we chose Decision Trees with feature set 1 that had an IoU of 83.2%. We also examined F1 score, Accuracy, Recall, and Precision. The classical ML models performed much better in terms of real-time computational speeds (FPS) but at the expense of considerable pre- and post-processing processing effort as well as extensive feature engineering. The CNN models provide an end-to-end solution for detecting drivable regions in snowy road surfaces by feeding in the raw image and predicting tire tracks without any feature engineering at the cost of slower real time compute speeds. The classical ML models do not handle variation and noise as well as the CNN models do. The CNN models offer a more mature solution to identify tire tracks in regions of snow-occluded lane lines. This study demonstrates that it is possible to detect drivable regions for specific scenarios of lane line occlusion due to snow using a single camera and existing technology. By enhancing image processing and tuning the CNN hyper-parameters, the results can be further improved. Additionally, having more data would significantly improve the CNN models and offer a more flexible model. Running the CNN models on a powerful computing machine would also result in faster compute speeds and allow data scalability. Future work to expand this study includes addressing other circumstances such as traffic lights, intersections, road curvature, turns, lane changes, active snowfall, and various lighting conditions. Overall, the problem of automated driving in adverse weather needs to be addressed in order to reduce the fatalities and economic costs that occur annually.

## References

1. Benson, A.J., Tefft, B.C., Svancara, A.M., Horrey, W.J.: Potential Reductions in Crashes, Injuries, and Deaths from Large-Scale Deployment of Advanced Driver Assistance Systems, pp. 1–8. Res. Brief (2018) [Online]. Available: https://trid.trb.org/view/1566022
2. Wenwen, S., Fuchuan, J., Qiang, Z., Jingjing, C.: Analysis and control of human error. Proc. Eng. **26**, 2126–2132 (2011)
3. Varghese, J.Z., Boone, R.G., et al.: Overview of autonomous vehicle sensors and systems. In: International Conference on Operations Excellence and Service Engineering, pp. 178–191 (2015)
4. Sternlund, S., Strandroth, J., Rizzi, M., Lie, A., Tingvall, C.: The effectiveness of lane departure warning systems-a reduction in real-world passenger car injury crashes. Traffic Inj. Prev. **18**, 225–229 (2017)
5. Kusano, K., Gabler, H., Gorman, T.: Fleetwide safety benefits of production forward collision and lane departure warning systems, SAE Int. J. Passeng. Cars - Mech. Syst. **7**(2), 514–527 (2014). https://doi.org/10.4271/2014-01-0166
6. Kusano, K.D., Gabler, H.C.: Comparison of expected crash and injury reduction from production forward collision and lane departure warning systems. Traffic Inj. Prev. **16**(Suppl 2), S109–14 (2015)
7. IIHS-real-world-CA-benefits.pdf, [Online]. Available: https://www.iihs.org/media/259e5bbd-f859-42a7-bd54-3888f7a2d3ef/e9boUQ/Topics/ADVANCED%20DRIVER%20ASSISTANCE/IIHS-real-world-CA-benefits.pdf

8. Advanced driver assistance systems: global revenue growth 2020-2023, Statista. https://www.statista.com/statistics/442726/global-revenue-growth-trend-of-advanced-driver-assistance-systems/ (accessed Apr. 24, 2023)
9. Jiménez, F., Naranjo, J.E., Anaya, J.J., García, F., Ponz, A., Armingol, J.M.: Advanced driver assistance system for road environments to improve safety and efficiency. Trans. Res. Proc. **14**, 2245–2254 (2016)
10. Asher, Z.D., Tunnell, J.A., Baker, D.A., Fitzgerald, R.J., Banaei-Kashani, F., Pasricha, S., Bradley, T.H.: Enabling prediction for optimal fuel economy vehicle control. Technical Report, SAE Technical Paper, 2018
11. Motallebiaraghi, F., Yao, K., Rabinowitz, A., Hoehne, C., Garikapati, V., Holden, J., Wood, E., Chen, S., Asher, Z., Bradley, T.: Mobility energy productivity evaluation of prediction-based vehicle powertrain control combined with optimal traffic management. Technical Report, 2022-01-0141, SAE Technical Paper, 2022
12. Kadav, P., Asher, Z.D.: Improving the range of electric vehicles. In: 2019 Electric Vehicles International Conference (EV), pp. 1–5 (2019)
13. Rabinowitz, A., Araghi, F.M., Gaikwad, T., Asher, Z.D., Bradley, T.H.: Development and evaluation of velocity predictive optimal energy management strategies in intelligent and connected hybrid electric vehicles. Energies **14**, 5713 (2021)
14. Mahmoud, Y.H., Brown, N.E., Motallebiaraghi, F., Koelling, M., Meyer, R., Asher, Z.D., Dontchev, A., Kolmanovsky, I.: Autonomous Eco-Driving with traffic light and lead vehicle constraints: An application of best constrained interpolation. IFAC-PapersOnLine **54**, 45–50 (2021)
15. How Do Weather Events Impact Roads? https://ops.fhwa.dot.gov/weather/q1_roadimpact.htm. Accessed 08 Oct. 2022
16. Gern, A., Moebus, R., Franke, U.: Vision-based lane recognition under adverse weather conditions using optical flow. In: Intelligent Vehicle Symposium, 2002, vol. 2, pp. 652–657. IEEE (2002)
17. Brandon, S.: Sensor fusion: a comparison of capabilities of human highly automated, [Online]. Available: http://websites.umich.edu/~umtriswt/PDF/SWT-2017-12.pdf
18. Lei, Y., Emaru, T., Ravankar, A.A., Kobayashi, Y., Wang, S.: Semantic image segmentation on snow driving scenarios. In: 2020 IEEE International Conference on Mechatronics and Automation (ICMA) (2020). https://doi.org/10.1109/icma49215.2020.9233538
19. Rawashdeh, N.A., Bos, J.P., Abu-Alrub, N.J.: Drivable path detection using CNN sensor fusion for autonomous driving in the snow. In: Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2021, vol. 11748, pp. 36–45. SPIE (2021)
20. Rjoub, G., Wahab, O.A., Bentahar, J., Bataineh, A.S.: Improving autonomous vehicles safety in snow weather using federated YOLO CNN learning. In: Mobile Web and Intelligent Information Systems, pp. 121–134. Springer International Publishing, New York (2021)
21. Goberville, N.A., Kadav, P., Asher, Z.D.: Tire track identification: a method for drivable region detection in conditions of Snow-Occluded lane lines. Technical Report, SAE Technical Paper, 2022
22. ZED 2 - AI stereo camera: https://www.stereolabs.com/zed-2/. Accessed 19 May 2022
23. Uddin, M.F., Lee, J., Rizvi, S., Hamada, S.: Proposing enhanced feature engineering and a selection model for machine learning processes. NATO Adv. Sci. Inst. Ser. E Appl. Sci. **8**, 646 (2018)
24. Duboue, P.: The Art of Feature Engineering: Essentials for Machine Learning. Cambridge University Press, Cambridge (2020)
25. Puget, J.-F.: Feature engineering for deep learning (2017). https://medium.com/inside-machine-learning/feature-engineering-for-deep-learning-2b1fc7605ace, Accessed 19 May 2022
26. Shetty, S.H., Shetty, S., Singh, C., Rao, A.: supervised machine learning: algorithms and applications, fundamentals and methods of machine and deep learning. Wiley, pp. 1–16 (2022). https://doi.org/10.1002/9781119821908.ch1

27. Chourasiya, S., Jain, S.: A study review on supervised machine learning algorithms. International Journal of Computer Science and Engineering. **6**(8), 16–20 (2019). https://doi.org/10.14445/23488387/ijcse-v6i8p104
28. Osisanwo, F.Y., Akinsola, J.E.T., Awodele, O., Hinmikaiye, J.O., Olakanmi, O., Akinjobi, J.: Supervised machine learning algorithms: classification and comparison. International Journal of Computer Trends and Technology (IJCTT). **48**(3), 128–138 (2017)
29. Seif, G.: Deep learning vs classical machine learning (2018). https://towardsdatascience.com/deep-learning-vs-classical-machine-learning-9a42c6d48aa, Accessed 19 May 2022
30. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2015)
31. Albawi, S., Mohammed, T.A., Al-Zawi, S.: Understanding of a convolutional neural network. In: 2017 International Conference on Engineering and Technology (ICET), pp. 1–6 (2017)
32. Why convolutional neural networks are the go-to models in deep learning, Analytics India Magazine, (2018). https://analyticsindiamag.com/why-convolutional-neural-networks-are-the-go-to-models-in-deep-learning/. Accessed 13 Feb 2022
33. Chatterjee, H.S.: A basic introduction to convolutional neural network (2019). https://medium.com/@himadrisankarchatterjee/a-basic-introduction-to-convolutional-neural-network-8e39019b27c4, Accessed 19 May 2022
34. Sankesara, H.: UNet (2019). https://towardsdatascience.com/u-net-b229b32b4a71, Accessed 19 May 2022
35. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: MICCAI (2015)
36. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: neural image caption generation with visual attention. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, (Lille, France), vol. 37, pp. 2048–2057. PMLR (2015)
37. Oktay, O., et al.: Attention U-net: learning where to look for the. Pancreas. **arXiv [cs.CV]** (2018). https://doi.org/10.7937/K9/TCIA.2016.tNB1kqBU
38. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition, arXiv [cs.CV], (2015). [Online]. Available: http://arxiv.org/abs/1512.03385
39. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the Loss Landscape of Neural Nets. **arXiv [cs.LG]** (2017) [Online]. Available: http://arxiv.org/abs/1712.09913
40. Iakubovskii, P.: segmentation_models: Segmentation models with pretrained backbones. Keras and TensorFlow Keras. Github. [Online]. Available: https://github.com/qubvel/segmentation_models,. Accessed 06 May 2022
41. Wikipedia Contributors: ImageNet (2022). https://en.wikipedia.org/w/index.php?title=ImageNet&oldid=1083632180
42. Kingma D.P., Ba, J. A.: A method for stochastic optimization, arXiv [cs.LG], (2014). [Online]. Available: http://arxiv.org/abs/1412.6980
43. Godoy, D.: Understanding binary cross-entropy/log loss: a visual explanation (2018). https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a, Accessed 12 Feb 2022
44. Yaqub, M., Jinchao, F., Zia, M.S., Arshid, K., Jia, K., Rehman, Z.U., Mehmood, A.: State-of-the-Art CNN optimizer for brain tumor segmentation in magnetic resonance images. Brain Sci. **10** (2020)
45. Classification: Accuracy, Google Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy, Accessed 24 Apr 2023
46. Duque-Arias, D., Velasco-Forero, S., Deschaud, J.-E., Goulette, F., Serna, A., Decencière, E., Marcotegui, B.: On power jaccard losses for semantic segmentation. In: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. SCITEPRESS - Science and Technology Publications (2021)