Deep Learning for Vehicle Recognition

Charles Galea

Table of Contents

Deep Learning for Vehicle Recognition

Abstract

We have generated an image recognition convolution neural network that performs well for identifying cars in a range of images. The network was trained against a dataset containing images of cars or cars involved in accidents. Additional random images were included to increase the generality of the classifier. Transfer learning utilizing the VGG16 CNN classifier did not significantly improve the performance of the network. Visual examination of feature maps pertaining to each filter for individual layers of the network indicated that the model generated in these studies retained significant information from the input images.

Introduction

CNN classifier

Convolutional layers

In a convolution neural network convolution layers act as image filters that extract feature (also called activation) maps that highlight the location and strength of detected features in an input image [1]. The stacking of convolutional layers allows for a hierarchical decomposition of the input image where the initial convolutional layers extract features that consist of combinations of lower-level features, such as features comprising multiple lines that form a shape while deeper layers extract higher-level features, such as faces, houses, cars, etc.

Dropout

Dropout is a regularization technique where randomly selected nodes are ignored during training [1]. Subsequently, their contribution to the activation of downstream nodes is temporally removed. During network learning some node weights tuned to specific features can 'memorize' images (often referred to as complex adaptations) leading to a fragile model too specialized to the training data. Randomly dropping out nodes during training avoids this problem by forcing other nodes to step in to compensate for the missing nodes. The network becomes less sensitive to specific node weights and results in a network that is more generalized and less prone to overfit the training data.

Data augmentation

Data augmentation is a method to generate more training data from an existing image dataset. Images in the training dataset are processed using a number of random transformations (e.g. rotation, shear, zoom, flip, etc). This produces a larger dataset, which ensures that the network does not see the same image twice and helps to build a more generalized model.

Transfer learning

Transfer learning is a highly efficient and effective method of leveraging a pre-trained network for classifying a relatively small image datasets [1, 2]. The pre-trained network has been previously trained on a large dataset such that the spatial feature hierarchy learned by the network can effectively act as a generic model for our smaller dataset. In this assignment we have used the VGG16 CNN classifier [3] that has previously been trained on the ImageNet dataset which contains 1.4 million labelled images and 1000 different classes.

Two methods can be used to make use of a pre-trained network: feature extract and fine-tuning [1]. Feature extraction consists of using the representations learned by a network to extract interesting features from a new sample. The features are used to train a new classifier. The level of generality of the representations extracted by specific convolutional layers depends on the depth of the layer in the model. Earlier layers extract local, highly generic feature maps (e.g. edges, colours and textures) while layers higher-up extract more abstract concepts (e.g. ear of elephant). Fine-tuning consists of unfreezing a few of the top layers of a frozen model base used for feature extraction and training both the newly added part of the model and these top layers.

Visualizing intermediate activations

Intermediate activations are useful for defining how successive convolutional network layers filter and transform the input images and determining the effect of individual convolutional filters [1]. Visualizing intermediate activation maps consists of displaying the feature maps that are output by various convolutional and pooling layers in the network. This provides an indication of how the input is decomposed into different features.

Problem Statement

The goal of this assignment was to develop a deep learning CNN-based image recognition model to classify images as containing damaged or intact cars.

Dataset

The cars dataset obtained from Prince Hemrajani (RMIT) and containing 3896 images of intact (1948 images) or damaged (1948 images) cars was used to train and validate a convolutional neural network (CNN) classifier.

Methodology

Data pre-processing

The dataset was filtered using a python script to remove corrupted images. The images were then processed to rescale the pixel values to between 0 and 1 and re-sized to a uniform number of pixels prior to feeding into the CNN classifier. All pre-processing steps were undertaken in the programming language Python (Python 3.6; Python Software Foundation, Wilmington, Del.;2009) using the package numpy (https://numpy.org/).

Data augmentation using the Keras ImageDataGenerator function was used to generate more training data by subjecting images in the existing dataset to a number of defined transformations.

Model training

The CNN-based classifier was trained and tested (validated) using 75% and 20% of the pre-processed dataset, respectively. The remaining 5% of the dataset was used to test the trained classifier.

Testing and debugging of the CNN classifier was performed on a local machine with a Windows 10 operating system, a six-core i7 2.7-GHz processor (Intel, Santa Clara, Calif) and 8 GB of RAM using a small subset of the entire dataset. Training and classification of the entire data was performed in either Google Colab using a dual Intel Xeon 2.2-GHz processor and a NVIDIA Tesla K80 graphical processing unit (Nvidia Corporation, Santa Clara, Calif) or on the Google Cloud Deep Learning platform using a NVIDIA Tesla P100 graphical processing unit. The packages Python 3.6.3 (https://www.python.org/), Tensorflow 1.10.0 (https://www.tensorflow.org/), Keras 2.2.2 (https://pypi.org/project/Keras/), CUDA 9.2.148.1 (https://developer.nvidia.com/cuda-zone) and cuDNN 7.2.1 (https://developer.nvidia.com/cudnn) were utilized for these deep learning studies.

The input shape was 128 x 128 x 3 with a binary output and the input images were standardized before being used for training. The CNN network consisted of a single input convolution layer, multiple hidden layers, a dense fully connected layer and an output layer. Adam (an adaptive learning rate optimization algorithm) was used for training the convolutional neural network [4].

Transfer learning using a pre-trained classifier

Feature Extraction

Weights and model architecture for the VGG16 CNN classifier previously trained on the ImageNet dataset were used to extract features for the car dataset. Part of the VGG16 CNN network excluding the final dense fully connected layers (the convolutional base) was used to train a smaller dataset. The VGG16 convolutional base was frozen and a new dense fully connected layer was added. The images are trained using the weights for the previously trained VGG16 convolutional base and the new dense layer [1].

Fine Tuning

Fine tuning is similar to feature extraction method except one or more of the layers of the frozen VGG16 convolutional base are unfrozen and the images are also trained on these layers [1].

Model interpretation and data visualization

Feature (or 'activation') maps for each convolutional and pooling layer in the network were extracted to gain insight into how the network derived its decisions. The feature map indicates how the input is decomposed by different filters learned by the network.

Results (please see appendices for code)

CNN model optimization

The pre-processed images were passed through a stack of convolutional layers containing filters with a small receptive field: 3 x 3 (the smallest size required to capture the various regions of the image – left/right, up/down and centre). Spatial pooling was carried out by max-pooling layers, performed over a 2 x 2 pixel window, following the convolutional layers.

The initial CNN configuration consisted of a convolutional input layer and a convolutional hidden layer followed by a dense (fully connected) layer with 128 channels and a soft-max output layer (configuration A in Table 1). The input and hidden layers are equipped with the rectification (ReLU) non-linearity [5]. The training data was processed in batches containing 32 64 x 64 pixel RGB images with binary labels.

The plots of model loss and accuracy (Fig. 1A) for the initial CNN network (config. A in Table 1) indicated that the model was overfitting. The model performed significantly better on the training data than the validation dataset (Table 3). The training accuracy increased linearly to reach a maximum of 100% within the first 10 epochs while the validation accuracy reached a maximum accuracy of about 75-80% within the initial few epochs and stalls. The validation loss reaches its minimum within the first 5 epochs while the training loss keeps decreasing to a minimum close to 0 within 10 epochs. The observed overfitting is likely due to the relatively low number of training samples (2890) in the dataset.

We initially determined the effect of increasing the image resolution. The size of the images (64 x64) used in the initial model was significantly less than the original image size (512 x 512) possibly resulting in a substantial loss of information. To examine this, we re-ran the model using images with a size of 128 x 128 pixels (configuration B in Table 1). The resulting loss and accuracy plots (Fig. 1B) again exhibited pronounced overfitting indicating that image size had no significant effect.

Several methods such as dropout, weight decay (L2 regularization) and data augmentation can be used to mitigate overfitting. Adding dropout regularization to each convolution layer and to the Dense layer, with a dropout probability of 25% (configurations C, D and F in Table 1), did not improve the model performance and overfitting was still significant in each case (Figs. 1C, 1D and 1F).

We explored the effect of inserting of an additional hidden convolution layer with dropout to determine what effect this would have on overfitting. However, this larger network (configuration E in Table 1) did not lead to reduced overfitting (Fig. 1E).

To determine whether the relatively small size of the dataset was contributing to the overfitting due to 'memorization of the data' we added data augmentation (shear, zoom and horizontal flip) to the model (configuration G in Table 1). This resulted in an improvement in model performance. Compared to the previous model, loss dropped from about 1.39 to 0.60 while accuracy increased from 0.76 to 0.81 (Fig. G and Table 3).

Changing of the number of validation steps from 50 to 32 (configuration H in Table 2) lead to negligible change in model loss or accuracy but decreased the time for training.

The addition of an another hidden convolutional layer with dropout (25% probability) (configuration L in Table 2) resulted in a gain in validation accuracy from 0.82 to 0.86 and decrease in loss from 0.48 to 0.33 (Table 3).

We also considered whether shuffling the dataset prior to training would improve the model (configuration M in Table 2). This resulted in a significant improvement with no apparent overfitting in the model (Fig. 1M). The training and validation curves were closely tracking each other while the final validation loss and gain (0.33 and 0.86, respectively) were similar to the training values (Table 3).

Finally, incorporation of a L1 regularization into the second convolutional layer lead to a further improvement in the performance of the model (configuration N in Table 2).

Table 1. CNN configurations. The depth of the configuration increases from left to right (A to G) as more layers are added (added layer are coloured in red). Conv2D – convolutional layer, Maxpool2D – pooling layer.

	CNN Network Configuration						
Parameters	**A**	**B**	**C**	**D**	**E**	**F**	**G**
Image Size	64 x 64	**128 x 128**	128 x 128	128 x 128	128 x 128	128 x 128	128 x 128
Epochs	50	50	50	50	50	50	50
Val steps	156	156	156	156	156	156	156
Conv layer	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D
Input_shape	(64, 64, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)
Activation	relu	relu	relu	relu	relu	relu	relu
Pooling	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D
Dropout			**Dropout (0.25)**	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
Conv layer					**Conv2D**	Conv2D	Conv2D
Activation					**relu**	relu	relu
Pooling					**Maxpool2D**	Maxpool2D	Maxpool2D
Dropout					**Dropout (0.25)**	Dropout (0.25)	Dropout (0.25)
Conv layer							
Activation							
Pooling							
Dropout							
Conv layer	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D
Activation	relu	relu	relu	relu	relu	relu	relu
Pooling	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D
Dropout				**Dropout (0.25)**	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
Flattening	Flattening	Flattening	Flattening	Flattening	Flattening	Flattening	Flattening
Dense layer	Dense(128)	Dense(128)	Dense(128)	Dense(128)	Dense(128)	Dense(128)	Dense(128)
Activation	relu	relu	relu	relu	relu	relu	relu
						Dropout (0.25)	Dropout (0.25)
Output layer	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)
Activation	softmax	softmax	softmax	softmax	softmax	softmax	softmax
Image aug	No	No	No	No	No	No	**Yes**

Table 2. CNN configurations. The depth of the configuration increases from left to right (H to N) as more layers are added (added layer are coloured in red). Conv2D – convolutional layer, Maxpool2D – pooling layer.

	CNN Network Configuration						
Parameters	**H**	**I**	**J**	**K**	**L**	**M**	**N**
Image Size	128 x 128	128 x 128	128 x 128	128 x 128	128 x 128	128 x 128	128 x 128
						shuffle	shuffle
Epochs	50	50	50	50	50	50	50
Val steps	**32**	32	32	32	32	32	32
Conv layer	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D	Conv2D
Input_shape	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)	(128, 128, 3)
Activation	relu	relu	relu	relu	relu	relu	relu
Pooling	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D
Dropout	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
Conv layer	Conv2D	Conv2D	**Conv2D**	Conv2D	Conv2D	Conv2D	Conv2D
Activation	relu	relu	**relu**	relu	relu	relu	relu
							L1 reg (0.0005)
Pooling	Maxpool2D	Maxpool2D	**Maxpool2D**	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D
Dropout	Dropout (0.25)	Dropout (0.25)		**Dropout (0.25)**	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
Conv layer					**Conv2D**	Conv2D	Conv2D
Activation					**relu**	relu	relu
Pooling					**Maxpool2D**	Maxpool2D	Maxpool2D
Dropout					**Dropout (0.25)**	Dropout (0.25)	Dropout (0.25)
Conv layer	Conv2D		Conv2D	Conv2D	Conv2D	Conv2D	Conv2D
Activation	relu		relu	relu	relu	relu	relu
Pooling	Maxpool2D		Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D	Maxpool2D
Dropout	Dropout (0.25)		Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
Flattening	Flattening	Flattening	Flattening	Flattening	Flattening	Flattening	Flattening
Dense layer	Dense(128)	Dense(128)	Dense(128)	Dense(128)	Dense(128)	Dense(128)	Dense(128)
Activation	relu	relu	relu	relu	relu	relu	relu
	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)	Dropout (0.25)
Output layer	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)	Dense (output = 1)
Activation	softmax	softmax	softmax	softmax	softmax	softmax	softmax
Image aug	Yes	Yes	Yes	Yes	Yes	Yes	Yes

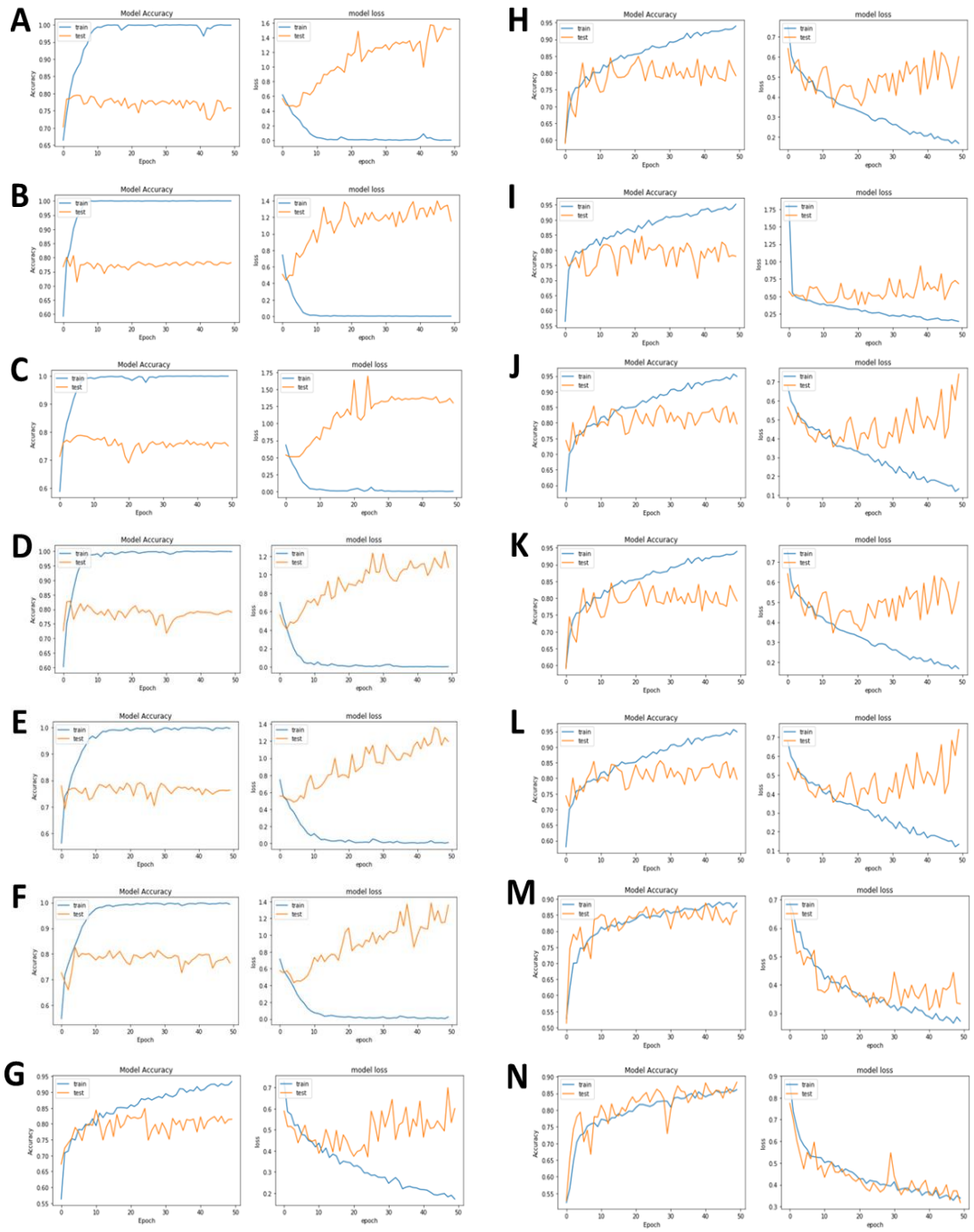


Figure 1. Training (blue line) and validation (orange line) accuracy (left panel) and loss (right panel) for the CNN model.

Table 3. Performance of various CNN models.

	CNN Network Configuration						
Performance	A	B	C	D	E	F	G
Train loss	0.0033	0.0013	0.0027	0.0036	0.0112	0.0244	0.1711
Train acc	0.9992	0.9996	0.9996	0.9988	0.9964	0.9936	0.9339
Val loss	1.5149	1.1535	1.3016	1.0771	1.1949	1.385	0.6003
Val acc	0.7577	0.7817	0.7507	0.7901	0.7628	0.7655	0.8147

Table 4. Performance of various CNN models.

	CNN Network Configuration						
Performance	H	I	J	K	L	M	N
Train loss	0.1679	0.1416	0.1327	0.1506	0.2614	0.2729	0.3387
Train acc	0.9399	0.9527	0.9491	0.9407	0.8878	0.8865	0.8613
Val loss	0.6016	0.6843	0.7411	0.4756	0.3266	0.3332	0.3176
Val acc	0.7916	0.7797	0.7966	0.8225	0.8654	0.8634	0.8833

We also checked to determine the performance of the CNN classifier for predicting the classes for images that it had not previously seen. Figure 2 shows predictions made by CNN network N for a set of random images taken from the test set. Overall, the CNN model performed well and was unsure for only one of the seven images.

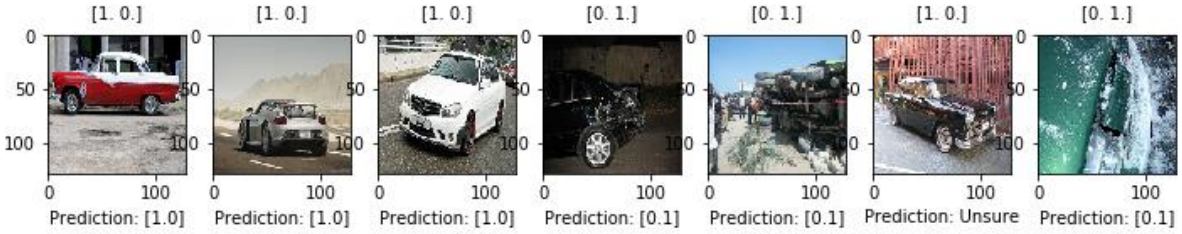


Figure 2. CNN model predictions for random images from the car dataset. The original classifications are shown above each image while predicted classification are below.

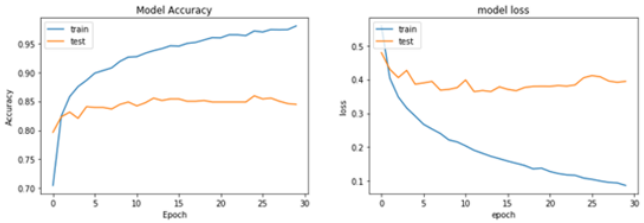
Transfer Learning

It has been shown that pre-trained networks can be utilized for deep learning on small datasets (also known as transfer learning). To test this, we used the large convolutional network VGG16 which was trained on the ImageNet dataset (1.4 million labelled images with 1000 different classes). There are 2 methods for leveraging pre-trained networks – feature extraction and fine-tuning.

Feature Extraction – Without data augmentation

We initially applied feature extraction, which uses representations learned by a previous network to extract meaningful features from our car dataset, without data augmentation (Fig. 3). The CNN model consisted of the convolutional base of the VGG16 model (containing the convolutional and pooling layers), which is the most generic and re-usable portion of the model, followed by a new densely connected classifier. The images were run through the convolution base of the VGG16 model using the weights previously acquired for this model and trained on the new densely connected part of the classifier (model F1). The model reached a validation accuracy of 84%, which was nearly was as good as model N (mentioned previously), however the model also exhibited overfitting (Fig. 3 & Table 5).

Figure 3. Transfer Learning with feature selection and no data augmentation. Training (blue line) and validation (orange line) accuracy (left panel) and loss (right panel) for the CNN model.



Feature Extraction – With data augmentation

To attempt to decrease overfitting we re-ran the feature extraction CNN model with data augmentation (i.e. rescale, shear, zoom and horizontal flip) (models F2). This resulted in a slight increase in the validation accuracy to 86% although overfitting was still observed (Fig. 4A). Increasing the number of augmentation parameters to include rotation, wide shift and height shift (model F3) however lead to a significant decrease in overfitting with a similar degree of accuracy (84%) (Fig. 4B & Table 5).

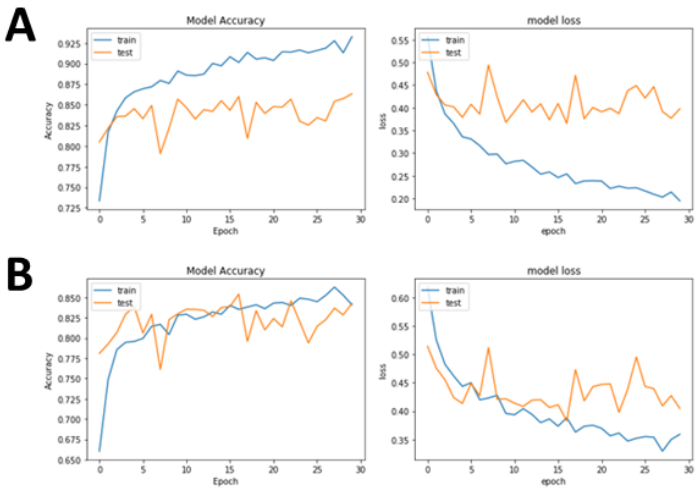


Figure 4. Transfer Learning with feature selection and data augmentation. (A) Data augmentation functions: (A) rescale = 1/255, shear = 0.2, zoom = 0.2, horizontal flip and (B) rescale = 1/255, shear = 0.2, zoom = 0.2, horizontal flip, rotation = 0.2, wide shift = 0.2, height shift = 0.2. Training (blue line) and validation (orange line) accuracy (left panel) and loss (right panel) for the CNN model.

Table 5. Performance of transfer trained CNN model using feature extraction.

	CNN Network Configuration		
Performance	F1	F2	F3
Train loss	0.0869	0.1974	0.3367
Train acc	0.9799	0.9313	0.8596
Val loss	0.3947	0.3978	0.3954
Val acc	0.8447	0.8633	0.8408

Fine Tuning

We were also interested in determining whether fine-tuning, another method for transferring the knowledge gained by a pre-trained network to a relatively small dataset, would improve performance. Unfreezing the final six convolutional layers of the VGG16 network did not lead to a significant improvement in the performance of the CNN model (Table 6).

Table 6. Performance of fine-tuned CNN model where a different number of convolutional layers of the VGG16 convolutional base have been frozen.

Performance	**CNN Network Configuration**				
	Conv4-1	Conv4-2	Conv5-1	Conv5-2	Conv5-3
Train loss	0.0069	0.0061	0.0080	0.0118	0.0284
Train acc	0.9968	0.9984	0.9978	0.9962	0.9928
Val loss	1.0976	0.7970	0.8805	0.6764	0.4565
Val acc	0.8601	0.8883	0.8601	0.8646	0.8716

Visualization of Activation Layers

To get an idea of how the images were processed by the CNN network that we had created we plotted the activations, which essentially defined the feature maps generated by individual filters within each layer.

The first layer appeared to be retaining the full shape of the car wherein some filters retain portions of the outline of the car (Fig. 4C) while others retain surfaces (Fig. 4D). Several filters are not activated and left blank. Overall, the first convolutional layer appeared to retain a significant amount of information from the image.

Activations for layers became more abstract and less interpretable when proceeding deeper into the network (Fig. 5). The deeper layers appeared to encode higher level information such as borders, corners and angles. These layers likely provide less information about the visual content of the image and more information related to the class of the object in the image.

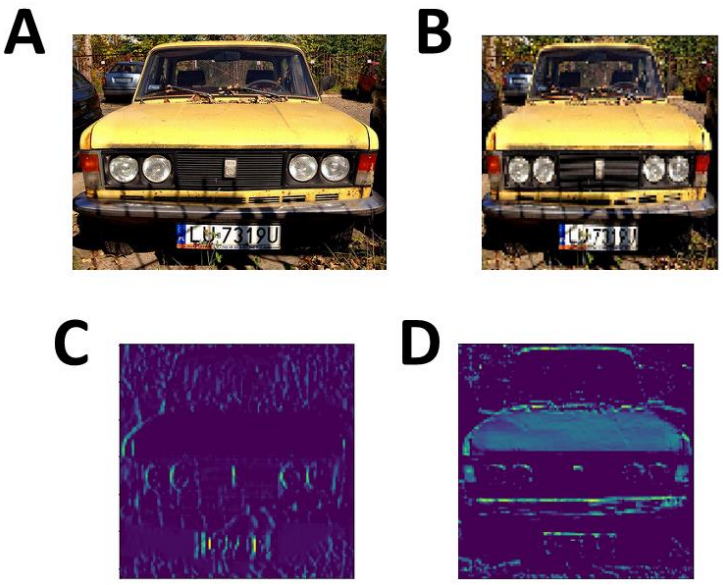


Figure 4. Activation map for CNN model. (A) Original and (B) processed image read into the network. (C) & (D) feature maps for different filters of a convolutional layer of the CNN model.

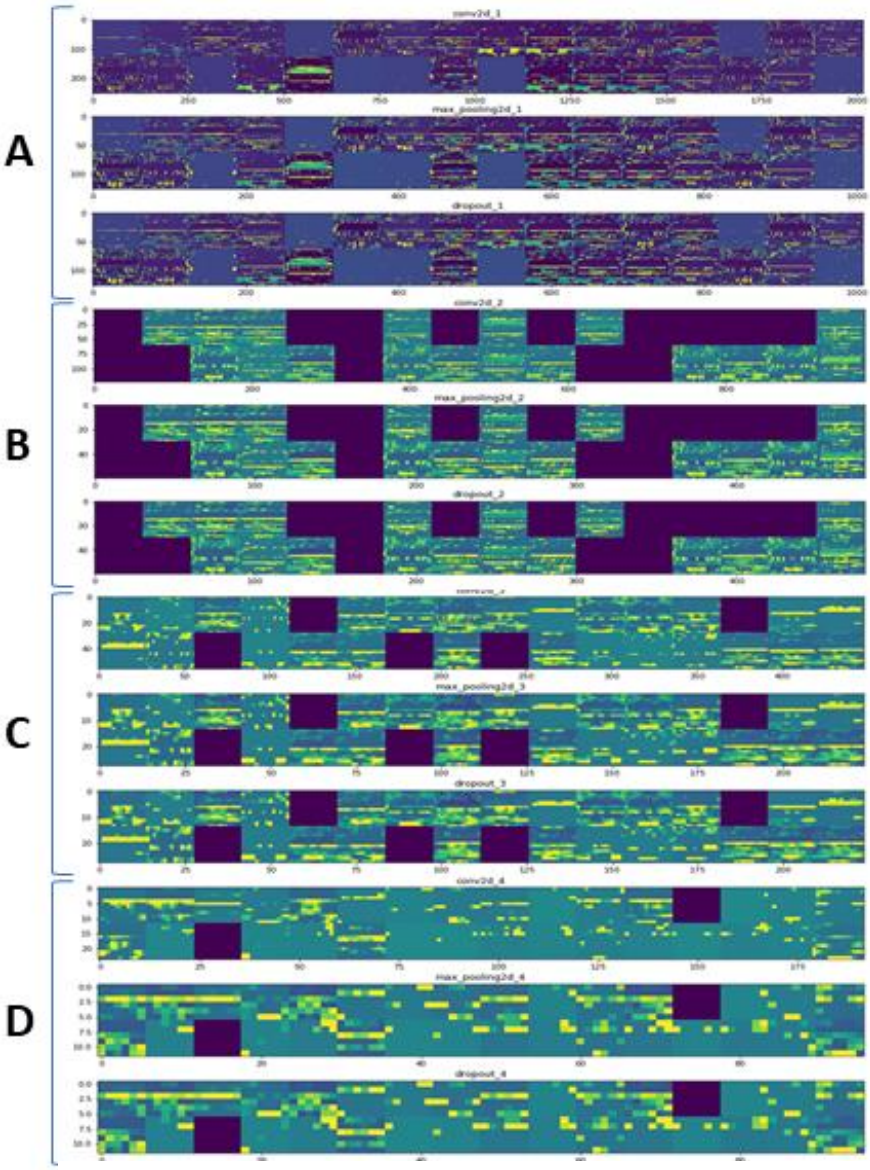


Figure 5. Feature maps for various convolutional layers of CNN model N. (A - D) Each layer is comprised of convolutional, maximum pooling and

Conclusions

The CNN network (model N) generated in these studies performed well with reasonably good accuracy and correctly predicting 6 out 7 random images taken from a test dataset. Due to difficulties in getting the images into a suitable format on the Google Colab server it was not possible to calculate additional measures of performance (e.g. confusion matrix, F1 score, precision and recall). Transfer learning using feature extract and utilizing the VG16 convolutional network did not perform well due to overfitting. Fine tuning produced a better result, similar to that of the generated CNN network (model N), with significantly less overfitting. Examination the feature maps suggested that this CNN network (model N) effectively captured information from images within the training dataset.

References

[1] F. Chollet, *Deep Learning with Python*. Manning Publications, 2017.

[2] C.-F. Blog, "Building powerful image classification models using very little data," *Keras Blog,* 2016.

[3] K. Simonyan and Z.-A. preprint arXiv, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.

[4] D. P. Kingma and B.-J. preprint arXiv, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural …,* 2012.