Preliminary Results

1 Problem Statement

The initial project discussed in Deliverable 1 is to create an app capable of classifying different road conditions accurately based on traffic images. For instance, if a user uploads a photo of some traffic, the model should be able to identify it as dry, wet, snowy, etc. as well as output other numerical, weather-related parameters such as temperature. The primary objective is to increase road safety by informing drivers about weather conditions so they can adjust their driving behaviour.

2 Data Preprocessing

The dataset employed for validation and testing is the same dataset suggested in the project proposal (Road CCTV images with associated weather data). The project proposal included several weather-related parameters to assess, but this preliminary results section will focus exclusively on road conditions since the other parameters will use a similar model. The main difference will be the data preprocessing beforehand. Out of the fifteen total columns of the image description, three were chosen (stationId, imgPath, and roadCondition). Other column metrics will be used in future steps. The stationId one is important because the dataset on the Harvard Dataverse website was separated into 556 different compressed files, each containing hundreds of images. The stationId name is also the name of its corresponding compressed file, so locating the image path relied on knowing the stationId number. Similarly, imgPath was used to fetch the images corresponding to the row metrics. The roadCondition column elements were used to create an array of labels to train and evaluate the model. The original dataset contains over 3.3 million images, which is too computationally expensive to train a model on. Instead, for the preliminary results, only around 5600 images were chosen from stationId numbers 10-0 and 100-0. While this is on the low end for CNN models, the model's accuracy and other metrics could be improved on by adding more images in later steps.

3 Machine Learning Model

The selected machine learning model is the same as the one suggested in the project proposal (CNN), because it is a strong model for image classification purposes. Since the course did not yet cover CNN models, I have decided to use an existing model (VGG16), and implement my own model later if time permits. I chose the VGG16 model because it is already trained on a large dataset (ImageNet), so training it on a small dataset will give more accurate results. Theoretically, its implementation should also be faster because of its pre-trained weights.

3.1 Frameworks and Tools

A high-level overview of the CNN model shows the architecture is composed of the VGG16 convolutional layers that form the base model, and a series of operations on the base model (for optimization purposes) that form the final model. The base model is composed of 13 convolutional layers and two fully connected layers (VGG16 usually has three, but the top layer has been removed because it pertains to the ImageNet dataset and is designed for 1000 classes, which is much more than the number of different road condition labels. The number of image pixels has also been reduced from the original VGG16 224 pixels to 128 pixels for computational efficiency. I used the tensorflow library to implement my base model (VGG16), its modifications (keras.layers), compilation (model.compile), and training (model.fit). I also used the cv2 library to read the images. I used a scikit-learn function to split the data into training and validation sets. I also used a tensorflow.keras function called to_categorical to convert arrays into matrices with binary inputs to enable one-hot encoding.

3.2 Model Setup and Training Decisions

The data was split into 20% for validation and 80% for training, so that the model has enough data to be properly trained while still setting aside enough data to gauge the model's performance accurately. A random state was also fixed so that the split remains the same for every run. As for regularization, a dropout layer was introduced and the VGG16's layers have been frozen. The dropout layer randomly deactivates half of the neurons during training, which means that the model cannot reply on specific neuron patterns and thus this layer prevents overfitting. Freezing

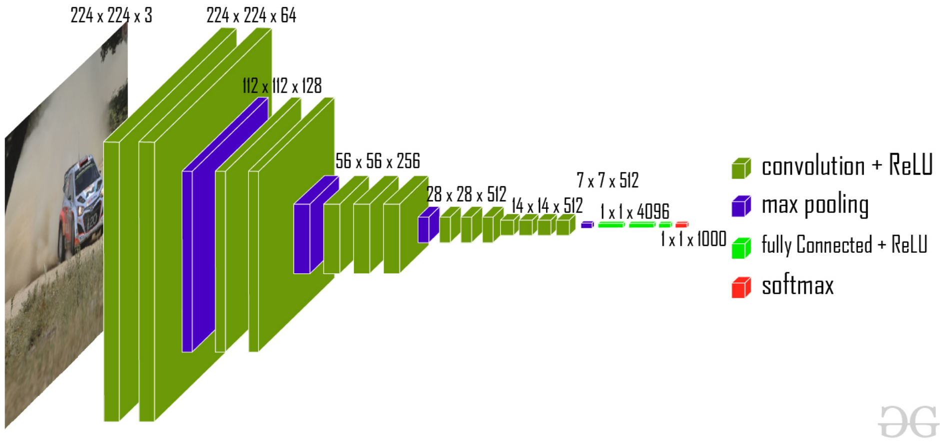

Figure 1: Architecture of the VGG16 Model (Source)

the VGG16 layers prevents the weights to be updated, which is important because if the weights were to change, the limited data might skew strong patterns and decrease the accuracy. The optimizer used is Adam because of its decent learning rate (0.001) and it is known to achieve fast convergence, which is convenient considering the low number of training epochs that are used. Since the model took such a long time to train, there was no change of hyper-parameters. Class weights do not need to be implemented for now because the dataset is fairly balance (at least, for the 10-0 and 100-0 folders). The number of epochs chosen for training is 10 because VGG16 is already pre-trained, so it converges faster. Additionally, a low number of epochs might prevent overfitting and takes less time to train, since training a single epoch took around 20 minutes.

3.3 Validation Methods

The data was tested using four metrics: training and validation accuracy, and training and validation categorical crossentropy loss. The accuracy simply refers to the proportion of correct classifications in the validation or training data set after each epoch. Categorical crossentropy is computed using a specific formula that compares how well predicted probabilities align with the actual distribution. Each of there four metrics was tested after every epoch. At this stage, the model is neither underfitting nor overfitting since accuracy increases while loss decreases steadily after each epoch iteration, which means the model is getting more precise, but there are also minor fluctuations in results which imply that the model is still generalizing and not overfitting. More importantly, the training and validation accuracies are similar, which suggests the model is not overfitting, or else it would have an increase of training accuracy while the validation accuracy would plateau.

3.4 Challenges

The main challenge with implementing this model was the time it took to train it. For 10 epochs, it look around three and three and a half hours to complete training, which made it difficult to fine-tune it since it was hard to gauge whether the model performed poorly because of the limited number of epochs or because of some different parameter. As such, the model testing step was severely limited, but using a GPU instead of a CPU should speed up the process in the future. Another challenge was to associate the image paths with their respective images because of inconsistencies in name formatting (for instance, some files are called ../2 while their image paths might indicate ../02). The solution for this was to sift through the image description csv file, pick the folders with the most images, and try to successfully load as many images as possible, which works because of the large number of images in the dataset as well as no presence of correlation between nonfunctional image paths and specific classes.

4 Preliminary Results

The full results can be seen in the code folder of the repository, in the Deliverable 2 notebook. For the preliminary results, the focus is on accuracy and loss, although the other metrics mentioned in the project proposal will be used in the future if necessary. The project proposal goal was to achieve an accuracy of 70%, which has been accomplished since the validation accuracy of the last epoch was 78%. There is still room for improvement, but this score confirms that the model performance is fairly effective. There can be tweaks with regards to batching, learning rates, increased dataset and epochs, testing with other baseline models, etc. to increase the accuracy score. Another result that has not been shown in the code repository is an attempt at data augmentation on the images

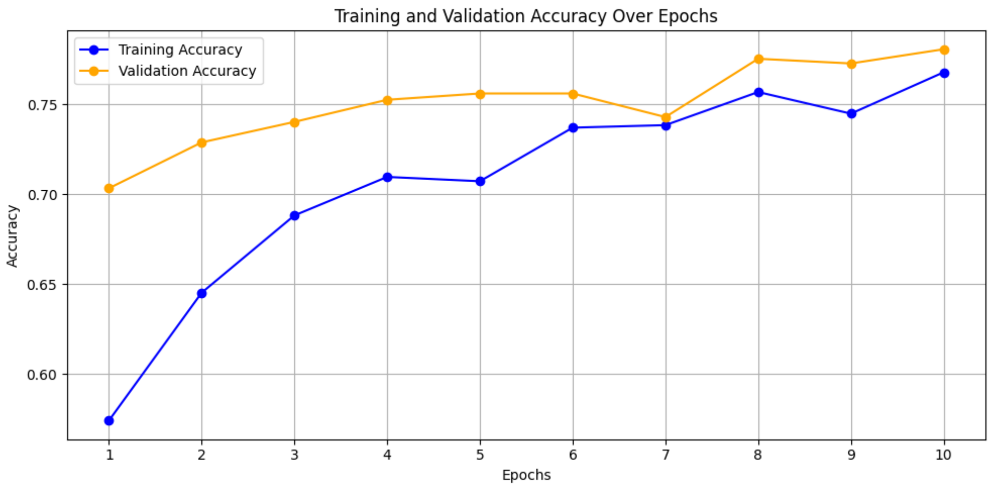

Figure 2: Training and Validation Accuracies of the Model over Number of Epochs

to increase the size of the dataset (and thus the accuracy), but the final training accuracy after ten epochs was around 66%, which is 10% lower than the score on the data without augmentation.

5 Next Steps

The next steps will be to try and refine the model; for instance, by fine-tuning parameters and testing out different implementations of CNN (or create a CNN model). Overall, the same model will be used (with a baseline model and some a customized layer), but the baseline model might change. Also, there needs to be many more images in the dataset, so another step would be to try to get to 20,000 images instead of 5,600. It is also extremely important to figure out how to speed up the training process if possible. Additionally, other columns that include weather metrics need to be included to be predicted alongside road conditions. Also, while training the model, a good idea would be to implement more metrics from the project proposal. After setting up a fine-tuned model with better training results, the next step would be to start coding the front-end and back-end of the web application.