Automated License Plate Recognition with Optical Character Recognition

Tyler Forgione

***Abstract*—In this project, two models were trained and combined: an automated license plate recognition model and an optical character recognition model. The license plate recognition model was trained on over 7000 images of street scenes containing zero or more license plates. This model was then used to find license plates in a dataset with images containing one or more license plates. Finally, these images were used to train a character recognition model used to read the license plates.**

I. Introduction

Automated License Plate Recognition

Automated license plate recognition (ALPR) is a mechanism used to detect and read license plates in images or video. There are a great number of applications, from speed cameras to parking lots to security of private property [1].

There are two main stages in ALPR: detecting the license plate and reading it. Detecting the plate is essentially locating an object with features specific to a license plate. These features include but are not limited to: colour, shape, ratio of height-width. These features are largely what separate license plates of different locations and from signs. For example, certain states have coloured license plates with white letters, whereas others have white plates with coloured letters. These are both license plates and the mechanism must be able to identify this.

There are numerous ways to detect license plates, from simple computer vision techniques (edge, colour, texture, character based) to deep-learning techniques like CNNs or YOLO [2]. I will be using YOLO to implement my ALPR mechanism.

After detection, the mechanism must be able to read the characters present on the license plate. To do this, I will be implementing something called optical character recognition (OCR). This is effectively a way for a computer to read machine-printed characters on an image [3].

A. YOLO

In this project, I train a model that can detect license plates as well as read them. First, I'll explain the base model that I'm using: You Only Look Once. YOLO (You Only Look Once) is a real-time object detection model [4]. Simply, this model splits an image into an SxS grid and each cell in that is responsible for detecting objects whose centre is inside it. Each cell will predict a bounding box for the item, confidence, and class probability.

What sets this model apart from tradition object detection models like R-CNNs is that it processes full images at once instead of going over them multiple times. This has two massive implications: YOLO is far faster than previous top object detection models, and it can encode context which may prevent mistakes that R-CNNs might make.

However, each cell in the image grid can only predict two bounding boxes which means small objects packed tightly together are suboptimal for the model to detect. Also, each cell can only predict one class but this does not matter for my purposes.

These benefits and limitations are directly related to license plate detection. First, if detecting license plates using a video stream, speed is of the utmost importance, which already makes YOLO a good model for this purpose. Even if using images instead of live video, if there is little to no tradeoff in accuracy, the faster model is typically chosen. Next, YOLO may use its deeper convolutional layers to encode the context of where license plates may lie. For example, since license plates are typically on cars, the model may recognize the structure of a car and increase its confidence of license plate location. I will be using a single class training set, so this may not matter; it may be that the model could recognize that license plate and car objects occur together but without a car class, the model may be more imprecise.

B. Optical Character Recognition

As mentioned previously, OCR is a way for a computer to read characters on an image. In this case, the ALPR mechanism will be reading characters off of license plates (either moving or still). The problem of character recognition has not been completely solved and while it has improved drastically, humans will continue to be better at this task [5].

Character recognition can be done in a number of ways, but the most typical is to use a CNN with CTC loss. This is known as a CRNN and has been used previously to attain great performance [6]. A CRNN is obviously a combination of a convolutional and recurrent neural network. In the context of OCR, the RNN used is Bidirectional Long-Term Memory (Bidirectional LSTM). Simply, images are passed sequentially through the CNN and the RNN and then trained using CTC loss.

Without getting into the in-depth mathematics, the reason this model works well for character recognition is due to its structure. First, CNNs are good at learning visual features (edges and contours) like simple cells in the early visual pathway. Local patterns recognition is a large part of being able to recognize characters (e.g: stroke orientation/length) but also provide some issues. For example, how can a CNN tell the difference between O and 0 (the letter O and the number

zero)? Moreover, how can a human do this? This is where the RNN comes in.

The bidirectional LSTM works both ways: it reads a sequence both forward and backwards. This allows the model to take in the context of the characters. The CNN passes a feature map to the RNN which act as evidence or context for positions in the image. This evidence can be thought of as probabilities. Essentially, what is the probability of character α in context Γ? In terms of license plates, a model may learn that characters are typically letters in the early positions and numbers in the later positions. Thus, the probability of a 1 may be lower than I at the start of the plate, but higher at the end of the plate.

Connectionist Temporal Classification (CTC) is a loss algorithm used to train deep-learning models, typically bidirectional RNNs. Effectively, CTC computes all possible alignments of input and output where the input is the predictions the RNN makes [7]. The algorithm will collapse multiple sequences of predictions down into one final output. The innerworkings of CTC are interesting and important for the model's output but are too complex for this paper. The 2006 Graves paper is a good explanation of how CTC works.

II. Methods

A. Training YOLO

The first thing to do in order to create an ALPR mechanism is to be able to detect license plates in a frame. YOLO is a well-known deep-learning model and has been updated numerous times with different weights. There are many versions of YOLO, from small to large, and version 11 is the latest one. This is the one I used simply because it is the most recent. Ultralytics notes that YOLOv11 is the model with the best performance relative to speed, which is a good enough reason for me. That is, time matters more to me than squeezing an extra few tenths of mAP@50–95 out.

In order to train this model, Ultralytics' Python package was used. This package contains methods to initialize and train Ultralytics models, as well as methods to predict bounding boxes and get data on validation sets. This model was trained on a set of just over 7000 images and validated on another 2000 images. Validation was performed three times in total with varying degrees of confidence. In this case, confidence describes what level of surety the model should have before predicting a bounding box. If confidence is set to 0.1, the model will not predict any bounding boxes if it is less than 10% sure it is a license plate.

The dataset this model was trained on was from Roboflow and was already prepared for YOLOv11 [8]. This means that a .yaml file was included along with the images which include the labels for the bounding boxes.

After training, the model's weights were saved and a finetuning session was performed with a learning rate of $1e^{-4}$ for 10 epochs. This was done to improve both mAP@50–95 as well as recall. Recall generally is quite important because misrepresenting license plates as background would make our OCR model not have a chance at reading the characters. In the case of this project, a better recall with worse precision is more welcome than the reverse.

B. Cropping

In order to perform OCR, the model will need to take as input images of license plates. This is obvious, seeing as it will be trying to recognize license plate characters. However, any background information can hinder the model's performance. Thus, the images need to be cropped based on the bounding boxes. If YOLO predicts a bounding box it thinks is a license plate, a zoom-in on that object is passed into the CRNN.

I did not realize that YOLO provided this function itself so I wrote a script using Python's cv2 package. This didn't take very long to write so it's not such a big deal.

C. Training the CRNN

The model I used was made using sequential CNN layers, RNN layers, and a fully connected layer. In total there are seven learnable layers in the CNN, four layers in the RNN (bidirectional LSTM, so 2*2), and one fully connected layer, making 12 total layers. Between the CNN layers, I added ReLU and max-pooling with a kernel size of 2 and stride of 1.

The dataset used to train this model was of 1700 annotated Indian license plates [9]. Unfortunately, while this dataset was annotated, it was not cropped and there were no bounding boxes around the license plates. Thus, I had to use my finetuned YOLO model on the dataset and save the cropped images. Note that when doing this, the model sometimes generates multiple bounding boxes for the same license plate. Each of these has its own issues: for example, the model may return a bounding box that is more zoomed in, cutting off the edges of letters. Manually going through these would've taken far too long, so a function was used to decide which crops were most in line with license plate dimensions.

After cropping, the images were then preprocessed. This meant taking each image and setting it to grayscale. Then, the images were each normalized to a height of 32 pixels. The point of this is to make each license plate image as readable and as similar as possible for the model. Colours may change how the characters are read and the height may affect which characters are which, since the pixel ordering may be changed slightly.

The preprocessed crops were then loaded into a DataLoader and split into train, validation, and test sets (80-10-10 split). The model was trained on the train set over 500 epochs using batch gradient descent (batch size = 64). The loss method used to optimize the model was CTC loss, described in the introduction. Two learning rates were used, $1\mathrm{e}^{-3}$ and $3\mathrm{e}^{-4}$, and early stopping was used only with the higher learning rate (patience = 30). The Adam optimizer was used in both training sessions as well as a scheduler with a factor of 0.5 and a patience of 20. This means that after 20 epochs of plateauing, learning rate gets cut in half.

III. Results

A. YOLO Training Results

YOLOv11 was trained for the first time with all the default hyperparameters over 100 epochs. The training stopped early at epoch 63, however epoch 61, seen in table I, showed the best mAP@50–95 and was thus chosen as the model for future use. Technically, it would have been possible to continue training until epoch 100 since the best model weights get saved. However, as mentioned above, recall was as important as mAP@50–95. Recall began fluctuating, so I stopped the model training and opted to finetune the model at a later time.

TABLE I: Detection performance of YOLOv11 on the validation set. The best model is selected based on the highest mAP@50–95.

Model	Epoch	Precision	Recall	mAP@50	mAP@50–95
YOLOv11	61	0.979	0.954	0.978	**0.718**

Over all epochs, the model consistently improved in most aspects. mAP@50–95 (figure 1) clearly improved until the end but was in a plateau in the sense that the increases were marginal.

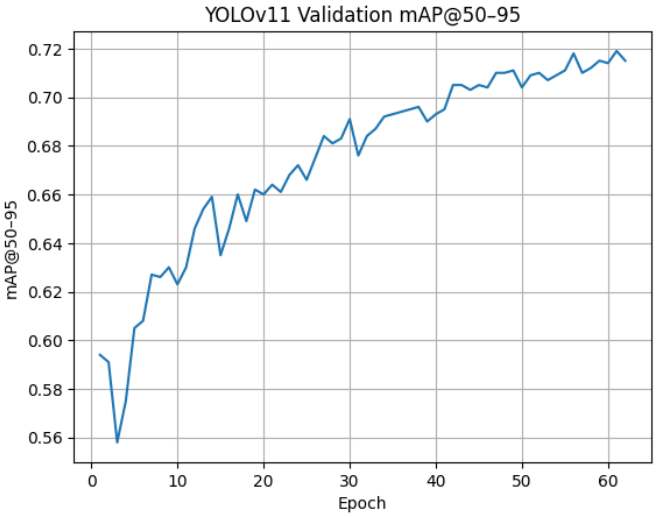


Fig. 1: Training mAP@50–95 of YOLOv11 over 62 epochs, showing stable convergence.

The box loss (figure 2) was still decreasing quite obviously even when the model was stopped early. Mainly, the reason for this is that box loss is not the metric that was most important. As stated, recall and mAP@50–95 were far more worthwhile to optimize for this project. Nonetheless, it is good to know that the model could've been quite a bit better than what it actually was.

Most important was recall, which began plateauing around epoch 40 (figure 3). This was the main reason for the early stopping. The figure shows the model gaining a lot of recall over its training and the recall chosen for the final model was

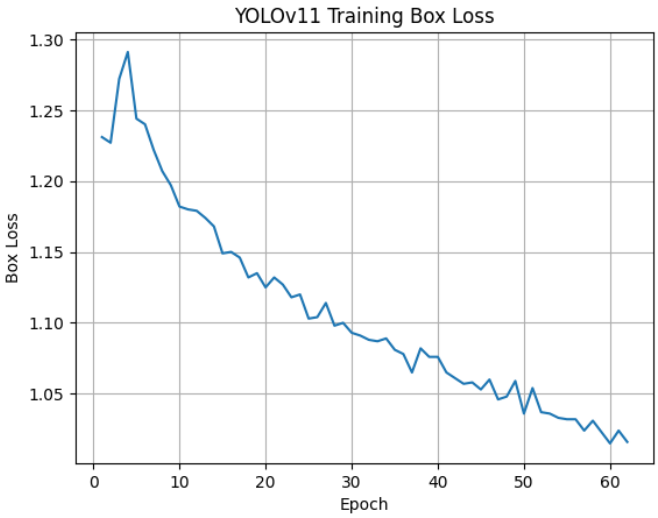


Fig. 2: Training box loss of YOLOv11 over 62 epochs, showing stable convergence.

0.954. That is, for every 1000 license plates shown, the model will only miss 46 of them.

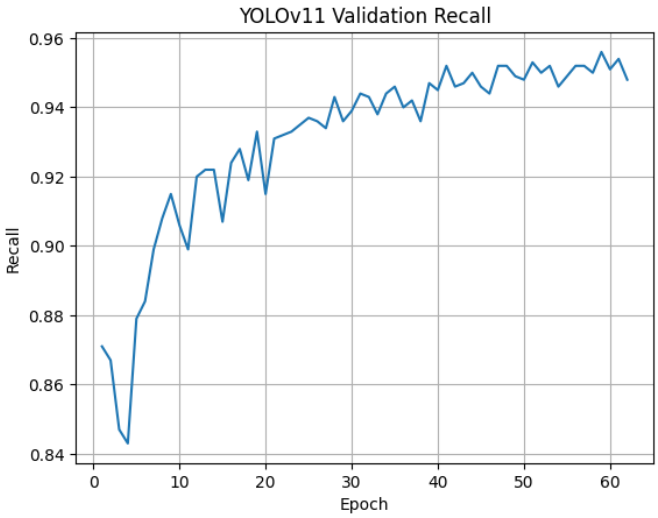


Fig. 3: Training recall of YOLOv11 over 62 epochs, showing stable convergence.

Given that the dataset had images of all kinds, meaning that the license plates were not always obvious, the model performed quite well. The next step, then, was to choose the correct confidence level based on performance on the test set. This was done by looking at the confusion matrices for three different confidence levels: 0.05, 0.10, and 0.20. Confidence is just how confident the model is that a bounding box is of the chosen class. Thus, a lower confidence threshold would lead to more objects being labeled as license plates overall. Then, precision would take a hit, since more non-license plates would be considered license plates. However, fewer license plates overall would be missed, implying higher recall. This

TABLE II: OCR evaluation metrics on the test set.

Metric	Value
Exact Match Accuracy	0.419
Character Accuracy	0.805
Character Error Rate (CER)	0.195

is exactly what is seen in the confusion matrices below (figure 4).

As expected, the lowest confidence level was best. It employed the best recall of the three confidence thresholds but the worst precision. With recall being more important, the balance achieved in the highest threshold was not worth the extra license plate misses.

B. CRNN Training Results

The CRNN training was done in two steps. First, because the dataset used was not annotated with bounding boxes (it only had license plate numbers), my YOLO model was used to predict bounding boxes. See the Methods section for the issues with this. Next, using the cropped bounding boxes, the CRNN was trained on the license plates my model found. Again, as mentioned in the Methods section, there were a number of issues with this.

The CRNN was first trained with a learning rate of 0.001 and then trained again newly with a learning rate of 0.0003. Early stopping was used the first time, but due to the high learning rate and the model's very late improvement in CTC loss, this early stopping hindered performance greatly.

In figure 5, the model converged extremely early. Unfortunately, this convergence was considered by my training algorithm to be the final one. Thus, the model ended with a CTC loss of around 2.5 and got no license plates fully correct on the test set.

Next, the model was freshly re-trained using the same hyperparameters except for a lower learning rate and a scheduler described in the previous section. This time, the training went far better and was manually stopped early when the plateau had gone on for too long. See figure 6 for the CTC loss curve.

Finally, this model was tested on the test set and performed quite admirably, though not perfectly. It got 41.9% of the license plates in the test set exactly right and, over all characters, correctly identified 80.5% of them.

Below is an example of the final OCR model being tested on a license plate crop that it gets correct as well as one that it gets incorrect. In figure 7, we see a license plate that the model correctly identifies as HP52B5958. This is particularly impressive seeing as the number 5, especially blown up like this, resembles the letter 'S'.

Next, the license plate in figure 8 was read as P8CA4850 by the model. This is wildly incorrect considering how much more readable the plate is than the previous one.

As a final example, the next license plate was completely incorrectly recognized by the model, meaning it got no characters right (figure 9). Here, the plate reads KLBOSS but the model read it as W2D.

IV. Discussion and Conclusion

A. Practicality of YOLO

YOLOv11 is a pretrained model, so my extra training could be considered finetuning. In any case, the model after training was extremely good with a chance of being even better. Since I was trying to maximize recall and not mAP@50–95, training was stopped when recall plateaud. However, as seen in the figures above, mAP@50–95 and box loss were continuing to increase and decrease, respectively, on the validation set.

Furthermore, once trained, the model was extremely quick at predicting bounding boxes. It only took 20 milliseconds per image during prediction. That is, in one second, the model was able to predict bounding boxes on about 3000 images. Thus, even on the fastest frame rate video, this model would be able to predict license plate bounding boxes in each frame.

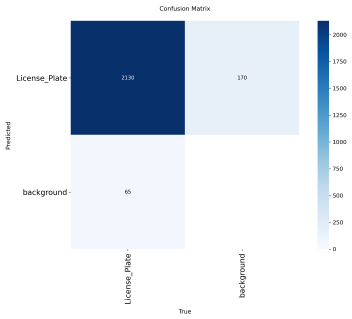
The live video usage is the most practical use of YOLO. If it were possible to trade some speed for accuracy, this would be an extremely important feature. Even if it took double the time to predict bounding boxes, an increase of just 3% recall would be worth it.

Perhaps surprisingly, it took the higher confidence threshold models (0.10 and 0.20) only 18 milliseconds per image during prediction. This seems like a miniscule, but over one minute, this comes out to an extra 300 images that the model can predict license plates for. This may be of use in the case that time is a factor, however it is unlikely to be needed. A model looking at license plates via live video would have other work to do like cropping, preprocessing, and OCR. These tasks take quite little time in general, so YOLO will generally never have trouble with lagging behind the frame stream.

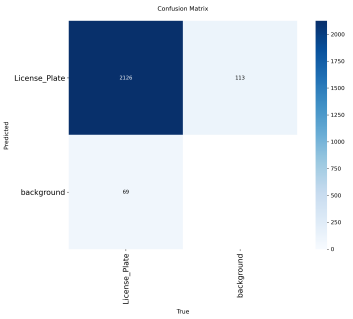
B. OCR Dataset Pitfalls

The dataset used for the Optical Character Recognition model was made up of Indian license plates. This is good because India is a very large country and made up of provinces. Thus, there are a number of very different license plate formats for the model to train on. That is, some are entirely on one line, some are broken up into two lines, some have spaces, and so on. This makes the model more robust, as it will make being able to read license plates from different parts of the world easier. For example, some European license plates are broken up into multiple lines, while some Canadian plates have spaces or very short sequences (vanity plates).

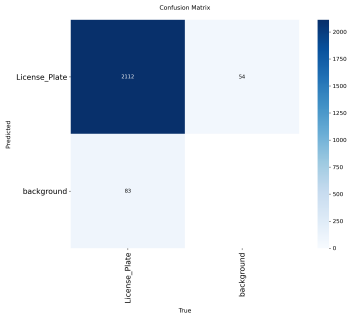
Unfortunately, India being a large country with many different plate types is a downside of the dataset. The dataset contained less than 2000 images and there weren't enough of certain types for the model to accurately understand the formatting. Looking back at the images above of license plates the model predicted incorrectly, one of them is extremely clear to read and it is obvious that the model was mistaken. However, the other one is completely unreadable. The top part could be any character, let alone two (KL). The bottom part could be entirely numbers or entirely letters (8055 vs BOSS), or even a mix of the two. The model cannot necessarily be blamed for this.



(a) Conf = 0.05



(b) Conf = 0.10



(c) Conf = 0.20

Fig. 4: Confusion matrices under different confidence thresholds.

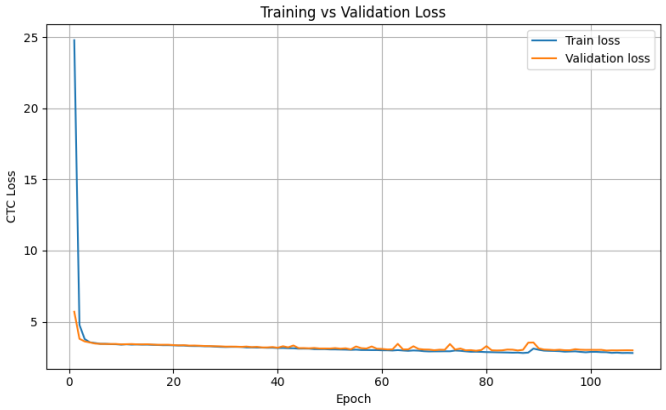


Fig. 5: Bi-CRNN training and validation CTC loss curve with learning rate 10^{-3} using early stopping.

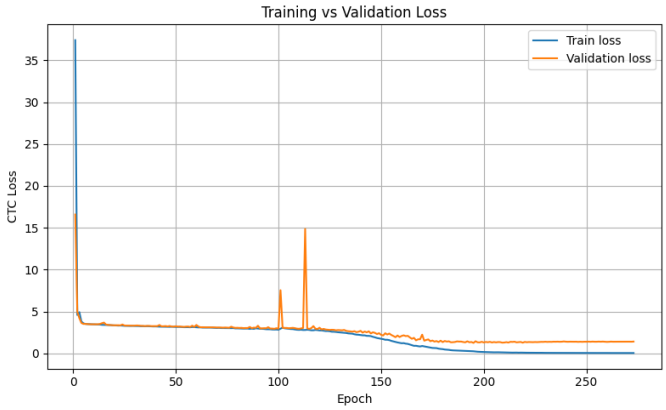


Fig. 6: Bi-CRNN training and validation CTC loss curve with learning rate $3 \cdot 10^{-4}$ using a scheduler (factor 0.5, patience 20) and no early stopping.

Getting a clearer image of this plate would involve getting a zoomed out image, which may cause further problems. Firstly, this would have to be a manual job, as YOLO saves images cropped at exactly the bounding box. So, the image to use would have to be either the image shown in this paper, or the full image of the car. In order to get something in between, I would have to manually edit the photo. Second, zoomed out images may contain the car's make and model. This gives the model something else to read, which makes its use in training a hindrance and its use in testing pointless. Last, using a zoomed out image leads to the model having to sift through clutter. This is the reason cropped images are used which contain only the license plate. The model should not need to fight the background scene in order to read the plate. That is what YOLO was used for during training and what it will be used for in practice.

C. Future Directions

In the future, I'd like to combine these models for live video streams. This would not only test YOLO's speed and ability

Fig. 7: License plate HP52B5958 used for model testing

Fig. 8: License plate DL8CX4850 used for model testing

to deal with highly cluttered images, but also test my OCR model's ability to deal with different license plates or even non-license plates. For example, YOLO counted street signs as license plates. Would the Bi-CRNN be able to read these, or would it fail because they are so different than the training data?

The training data used for the CRNN was made up of Indian license plates, which are starkly different to license plates in North America. Thus, getting another annotated dataset with North American plates to finetune the model on would be optimal. Importantly, Indian license plates have symbols at times on the side or top of the plate but no words. North American ones have slogans for the state/province as well as the name of that state/province. Thus, the next question to ask with regards to North American plates is whether the OCR model would fail at reading the plate characters because of the slogan and province name.

D. Conclusion

The two models trained performed extremely well considering the lack of data, specifically for the OCR model. YOLOv11 is a fast, performant model and can be finetuned to the project's specifications with only a few hours of training on GPU. The bidirectional CRNN used for character recognition felt the effects of the dataset being small but still did a good job on unseen data, all things considered. It boasted an accuracy of about 42% on license plates and 80% on characters.

As a learning project, this one was quite enjoyable. I learnt a lot about how computer vision techniques can be utilized to recognize not only large objects but also numerous small objects as well as characters. Obviously, a lot of it is pattern recognition, but the fact that these models can tell apart 0 and O or 5 and S is quite impressive.

I also learnt about CTC loss, which I had not heard of prior to this project. In the future, I'd like to use CTC loss in a speech recognition model.

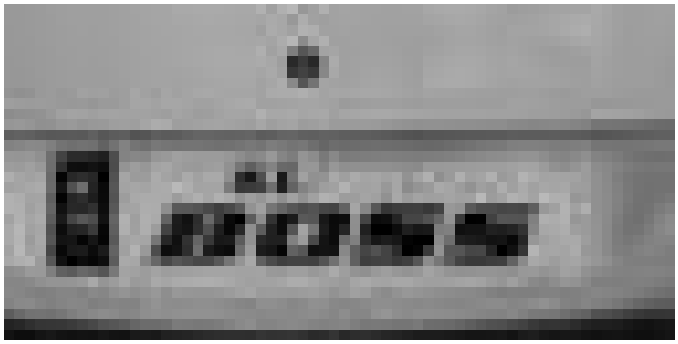


Fig. 9: License plate KLBOSS used for model testing

REFERENCES

[1] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, "Automatic license plate recognition," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 1, pp. 42–53, 2004.

[2] J. Shashirangana, H. Padmasiri, D. Meedeniya, and C. Perera, "Automated license plate recognition: A survey on methods and techniques," *IEEE Access*, vol. 9, pp. 11 203–11 225, 2021.

[3] L. Eikvil, "Optical character recognition," *citeseer. ist. psu. edu/142042. html*, vol. 26, 1993.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," May 2016. [Online]. Available: https://arxiv.org/abs/1506.02640

[5] N. Islam, Z. Islam, and N. Noor, "A survey on optical character recognition system," 2017. [Online]. Available: https://arxiv.org/abs/1710.05703

[6] I. Campiotti and R. Lotufo, "Optical character recognition with transformers and ctc," in *Proceedings of the 22nd ACM Symposium on Document Engineering*, ser. DocEng '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3558100.3563845

[7] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 369–376. [Online]. Available: https://doi.org/10.1145/1143844.1143891

[8] R. U. Projects, "License plate recognition dataset," https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e, apr 2025, visited on 2026-01-09. [Online]. Available: https://universe.roboflow.com/roboflow-universe-projects/license-plate-recognition-rxg4e

[9] C. M. Bhuma, V. M. S. N. P. K. CH, B. Katakam, V. V. Bethala, S. Vatada, and S. Kagitha, "Number plate number identification dataset," 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13954136