# Machine Learning Exercise 3
# Topic 3.2.1: Image classification - Feature Extraction & Shallow vs. Deep Learning

## 1 Datasets

### 1.1 CIFAR-10

The CIFAR-10 dataset is a widely used benchmark for image classification tasks. It consists of 60,000 color images of size 32×32 pixels, divided into 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset is split into 50,000 training images and 10,000 test images. Due to its low resolution and significant variation within each class, CIFAR-10 is commonly used to evaluate and compare both traditional machine learning methods and deep learning models.

We downloaded the data from this site[1] (*CIFAR-10 python version*) .

### 1.2 GTSRB

The German Traffic Sign Recognition Benchmark (GTSRB) is a real-world dataset for traffic sign classification. It contains over 50,000 images covering 43 different traffic sign classes, collected under diverse real-world conditions. The images show significant variation within each class due to changes in lighting, scale, rotation, motion blur, and background complexity. Image resolutions vary, and the dataset reflects realistic class imbalance. GTSRB is widely used to evaluate the robustness of traditional machine learning approaches and deep learning models, particularly in applications related to autonomous driving and intelligent transportation systems.

For this dataset, we visited this site[2], where we downloaded the following folders:

- *GTSRB_Final_Training_Images.zip*: contains train images with their labels
- *GTSRB_Final_Test_Images.zip*: conains test images with a csv file. However, this csv file does not contain the labels. For this reason, we do not use it.
- *GTSRB_Final_Test_GT.zip*: contains the labels for the test images.

## 2 Methodology

### 2.1 General

We used Python for the implementation as the main development tool including several libraries / frameworks.

For the traditional machine learning approach, we first focused on extracting meaningful features from the raw data. We employed 3D Color Histograms as well as SIFT descriptors combined with a Bag-of-Visual-Words (BoVW) approach. These features served as input for two different classifiers to establish a baseline: Logistic Regression as a linear model and Random Forest as a representative ensemble method for non-linear relationships. To optimize model configurations and hyperparameters, we additionally used GridSearch in combination with Cross-Validation.

Since the data in both datasets was already splitted into train and test set, we did not have to do a separation. As common, we used the train set to train the DL models and then evaluated the corresponding performances of the different models on the test sets. In order to measure performance, we primarily relied

[1] https://www.cs.toronto.edu/ kriz/cifar.html
[2] https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html

on the (balanced) accuracy and loss metrics (also other metrics reported, look at next paragraph). Besides, we also tracked runtime in terms of training and testing for each model configuration.

Since training a model can last very long, we tried to track any possible information during the train step. For each differently configured model, we stored values such as total train time, accuracy, loss. We also reported Precision, Recall and the resulting F1-score per class regarding the last epoch of a trained model. Also nice to see, we stored the loss and accuracy evolution for each model during train and test processes. A plot showing the confusion matrix is also included.

Scripts are also included in the submission, which are configurable via a command-line options for setting model parameters. How to use the command-line option is described in detail in the submission files.

In order to avoid huge numbers of calculations that cause many hours of training sessions, we input images in the 32x32 format to the DL models. However, our implementation also supports 64x64 inputs.

In the following, the chosen shallow ML algorithms and DL models are introduced.

## 2.2 ML approaches

### 2.2.1 Logistic Regression

Since Logistic Regression is a widely used baseline classifier, we applied it to both the CIFAR-10 and GTSRB datasets. The model learns a linear decision boundary during training and outputs class probabilities. To support classification across multiple classes, the model is extended using a one-vs-rest approach. Two hyperparameters are considered: the regularization parameter (C), which controls the trade-off between model complexity and generalization, and the penalty parameter, which defines the type of regularization applied.

Logistic Regression is computationally efficient and easy to interpret.

### 2.2.2 Random Forest

Since Random Forests are robust ensemble classifiers that handle complex patterns well, we applied them to both the CIFAR-10 and GTSRB datasets. Unlike Logistic Regression, which learns linear decision boundaries, Random Forests build multiple decision trees and combine their predictions through majority voting, making them fundamentally different and well-suited for capturing non-linear relationships. Key hyperparameters are explored: the number of estimators (n_estimators), which defines how many trees are built, and the maximum depth (max_depth), which controls tree complexity. Random Forests are computationally more expensive than Logistic Regression but offer stronger performance through their ability to model complex, non-linear patterns.

## 2.3 DL approaches

### 2.3.1 Simple LeNet5 Architecture

As the first model, we wanted to use a simple one. For this reason, we chose the LeNet5 model, as it was also mentioned in the assignment. It contains convolutional and pooling layers followed by fully connected layers. Since there are many implementations provided in the internet, we re-used some implementations, referred to in the code. Although it is not included in the origin model, we also added the dropout option to inspect the impact of dropout. We varied the following parameters to look at different scenarios:

- Dropout: 0.0, 0.2, 0.5
- Number of Epochs: range: 1, 5, 10, ... , 90 (depending on dataset / relevance)
- Augmentation: Yes, No

Many other parameters, which are configurable, have been set to default values:

- Activation Function: tanh

- Image Size: 32x32
- Optimizer: Adam
- Learning Rate: 0.001

#### 2.3.2 ResNet18 Architecture and Transfer Learning

As our second model, we used ResNet18, a more modern CNN architecture compared to LeNet5. The key feature of ResNet is its residual connections. This feature addresses the vanishing gradient problem in deep networks. Basically, they allow the network to skip layers; instead of learning complete transformations, each layer learns only the residual difference.

A major advantage of ResNet18 over LeNet5 is its capability for transfer learning. This allows us to use ImageNet pretrained weights as a starting point. We tested both training from scratch and using ImageNet pretrained weights. Additionally, we have the option to freeze the backbone. When activated, the convolutional layers are frozen and not updated during training (advantage: significant resource savings). In this configuration, the frozen backbone provides learned feature representations from ImageNet, while only the final classification layer is trained. This is why freezing doesn't make sense when we're not using transfer learning – freezing random weights provides no benefit.

We also added Dropout for the final layer to combat potential overfitting problems. Otherwise, we used similar parameters as with LeNet.

We varied the following parameters during our experiments:

- Dropout: 0.0, 0.2, 0.5
- Number of Epochs: 20
- Augmentation: Yes, No

Other parameters were kept at default values:

- Image Size: 32×32
- Optimizer: Adam
- Learning Rate: 0.001
- Activation Function: ReLU (ResNet default)

### 2.4 Data Augmentation

We applied dataset-specific augmentation to the training set only to improve model generalization and robustness:

- **GTSRB:** We utilized ColorJitter (brightness/contrast ±15%, hue ±2%), RandomRotation (±8°), and RandomAffine (translate ±3%, scale 0.95–1.05). These transformations simulate realistic lighting conditions and camera angles while preserving sign semantics, ensuring that color and shape remain recognizable.
- **CIFAR-10:** We applied RandomCrop (32×32, padding=4) and RandomHorizontalFlip ($p = 0.5$). These represent standard augmentations for natural images, introducing shift and flip invariance to the model.

## 3 Results

In this section, we will compare the different approaches across different aspects.

## 3.1 ML approaches

### 3.1.1 Feature Extractions

Before training our classifiers, we transformed the raw image data into feature vectors using two distinct approaches -3D Color Histogram and SIFT + Bag of Visual Words

- **3D Color Histogram:** We computed a joint histogram across the three color channels using **8 bins** per channel. This results in a feature vector that captures the global color distribution of an image but discards all spatial information.
- **SIFT + Bag of Visual Words:** We used the Scale-Invariant Feature Transform (SIFT) to detect local keypoints (edges, corners, textures). To aggregate these variable-length descriptors into a fixed-size vector, we applied the Bag of Visual Words (BoVW) approach. We used K-Means clustering to build a vocabulary of 100 visual words (clusters). Each image is then represented as a histogram of these visual words.

We analyzed the time required to extract these features for both the training and testing sets. Figure 1 illustrates the results for both datasets (GTSRB and CIFAR-10).

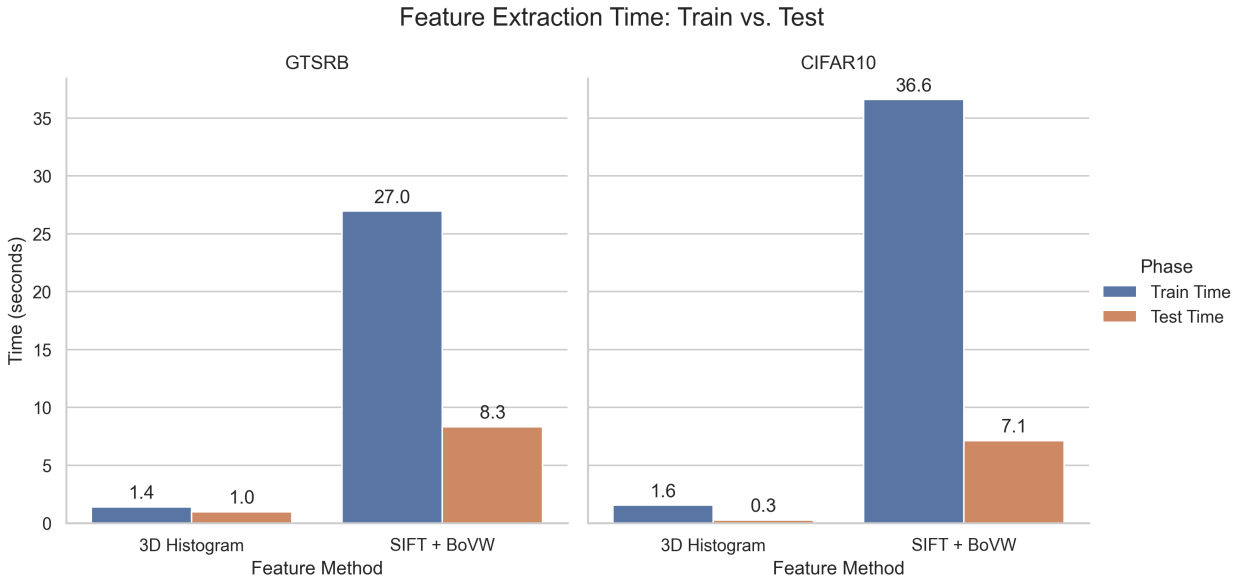


Figure 1: Runtime comparison of feature extraction methods.

On both datasets the histogram method is much faster than SIFT.

### 3.1.2 Random Forest

To evaluate the effectiveness of our extracted features, we first trained a Random Forest classifier. We performed a GridSearch to optimize the hyperparameters, using 3-Fold Cross-Validation to ensure the robustness of our results.

We explored the following hyperparameter space:

- **Number of Estimators (*n_estimators*):** 30, 60, 90
- **Maximum Depth (*max_depth*):** 5, 10, 15

Figure 2 displays the Mean Cross-Validation Test Score for both datasets.

For GTSRB, we observe a clear pattern across both features. Performance improves consistently as the maximum tree depth increases. Throughout all depth configurations up to the maximum, SIFT outperforms

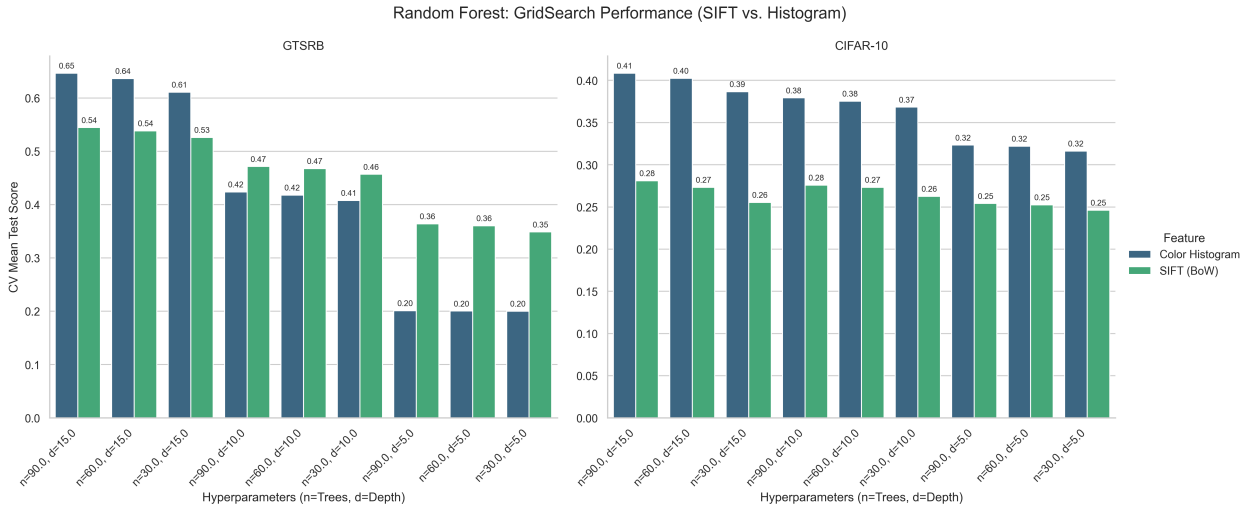


Figure 2: Random Forest GridSearch Performance.

Color Histogram. For CIFAR-10, however, the pattern is less pronounced. Color Histogram shows only marginal improvement with increasing tree depth, while SIFT performance remains nearly unchanged. Here, Color Histogram achieves better results across all configurations.

We also analyzed the time required to fit the Random Forest models for each fold, as shown in Figure 3.

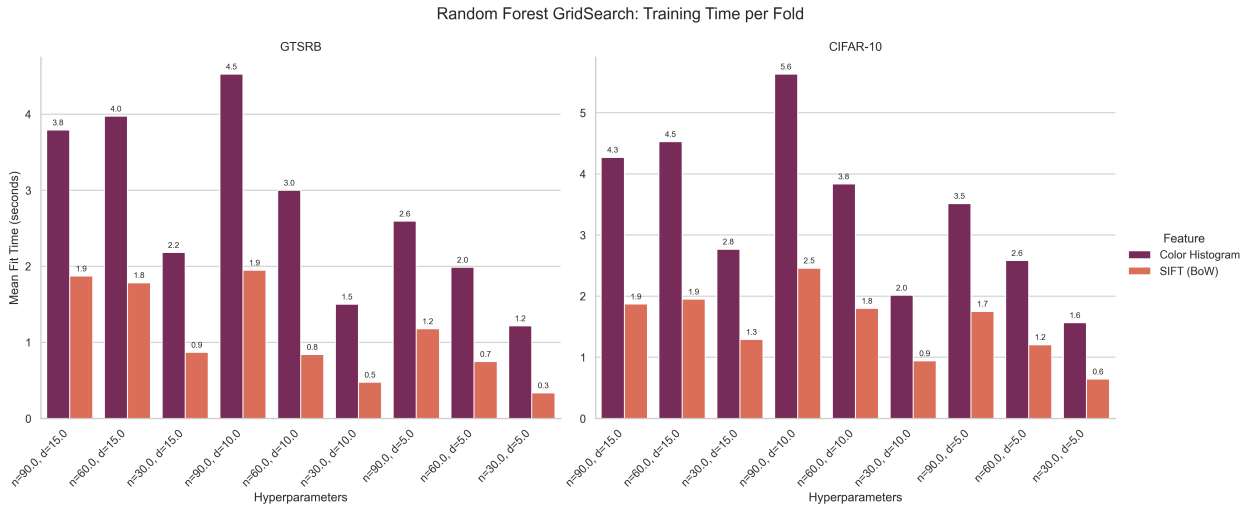


Figure 3: Training time per fold.

Here, we observe an interesting reversal compared to the feature extraction phase. Although SIFT was much slower to extract, it is significantly faster to train than the Color Histogram.

To better understand why the Color Histogram consistently outperformed the SIFT approach, we generated confusion matrices for the best performing Random Forest models from our GridSearch. Which we can see in Figure 4. This is the confusion matrix for the CIFAR-10 dataset.

In the left plot, we can see that the diagonal is significantly darker than in the right plot. Especially Class 3 (cat) and Class 4 (deer) did not perform very well in the SIFT plot. We observe the same result in Figure 2.

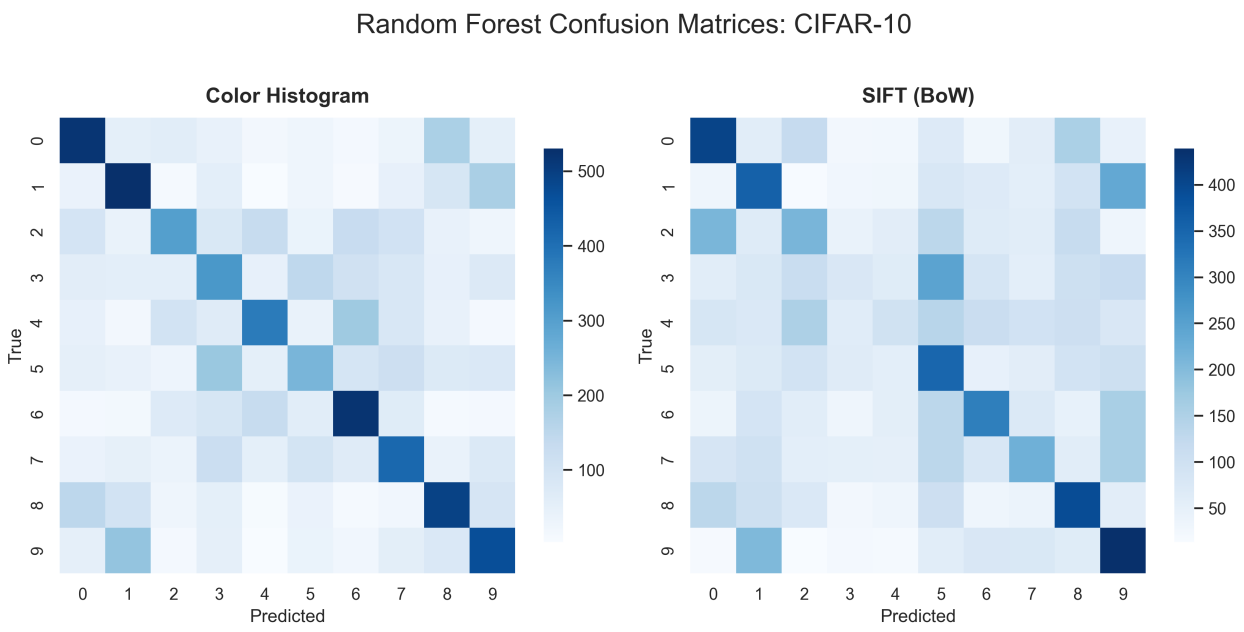


Figure 4: Confusion Matrices for the best Random Forest models on CIFAR-10.

Figure 5 displays the results for the GTSRB.

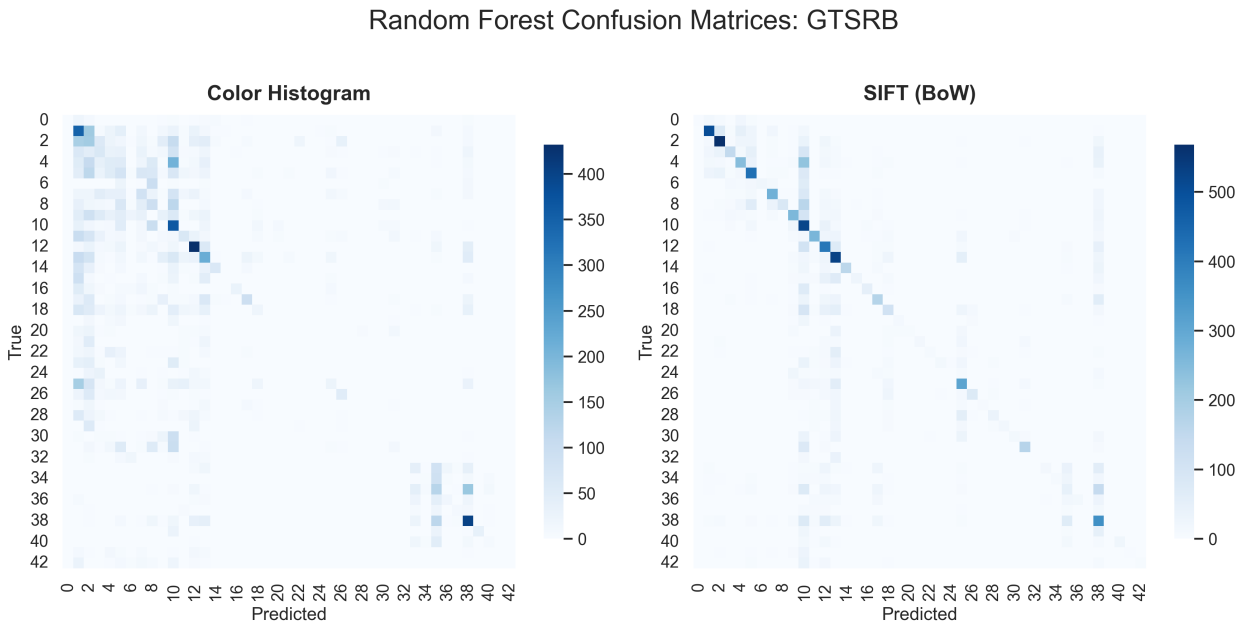


Figure 5: Confusion Matrices for GTSRB (43 Classes).

In Figure 5, we now see that SIFT beats the histogram. The diagonal is much more distinctive, and in the top-left corner of the Color Histogram, there are distinct patches of confusion. This time, this is a contradictory result to Figure 2, which claimed that the Histogram is better than SIFT.

In Table 1, we can see the reason why. While the Color Histogram achieved a very high Cross-Validation Score (0.6464) on GTSRB, the final Test Accuracy dropped massively to 0.2313. This indicates severe overfitting—the model memorized the color distributions of the training set but failed to generalize. The

SIFT model was much more stable (CV: 0.54, Test: 0.47).

Table 1: Summary of Random Forest Performance (Test Set vs. CV Score). Comparison of GTSRB and CIFAR-10.

| Feature | CV Score | Accuracy | F1 Score | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|
| **GTSRB** | | | | | |
| Color Histogram | 0.6464 | 0.2313 | 0.1452 | 3.79 | 0.0711 |
| SIFT (BoW) | 0.5444 | **0.4721** | **0.3393** | 1.87 | 0.0485 |
| **CIFAR-10** | | | | | |
| Color Histogram | 0.4085 | **0.4196** | **0.4159** | 4.27 | 0.0408 |
| SIFT (BoW) | 0.2811 | 0.2866 | 0.2738 | 1.87 | 0.0248 |

### 3.1.3 Logistic Regression

Subsequently, we evaluated a Logistic Regression classifier to determine how a linear model handles the extracted features. Similar to the Random Forest approach, we employed GridSearch with 3-Fold Cross-Validation.

The hyperparameter search space focused on regularization strategies:

- **Penalty:** None, Lasso (L1), Ridge (L2), ElasticNet
- **Inverse Regularization Strength ($C$):** 0.1, 1, 10

Figure 6 illustrates the Mean Cross-Validation Test Score.

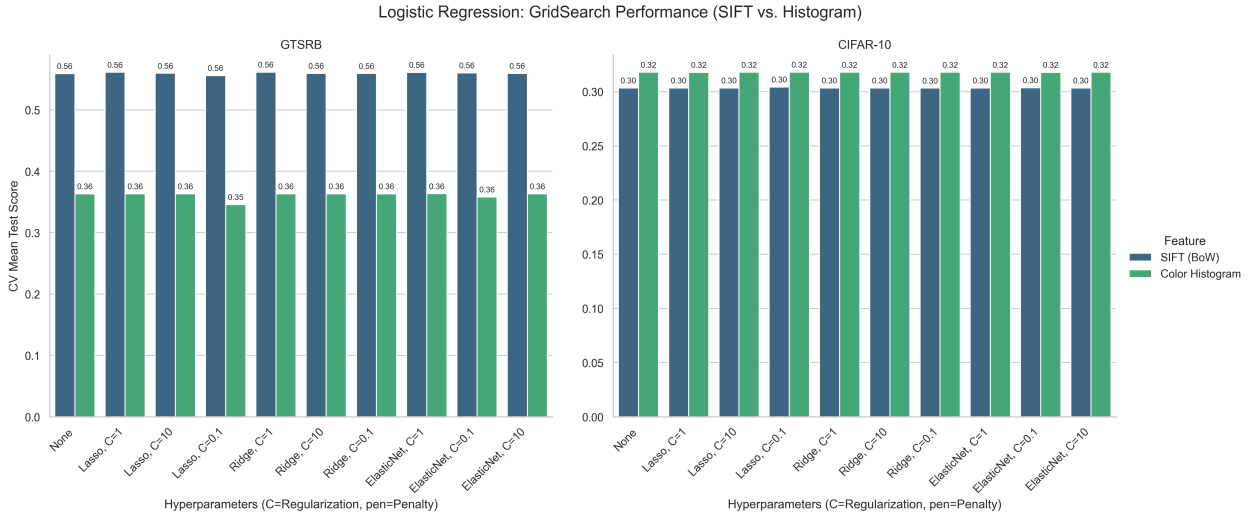


Figure 6: Logistic Regression GridSearch Performance (SIFT vs. Histogram).

The results for Logistic Regression reveal an contrast to the Random Forest results. SIFT is on the GTSRB set much better than histogram. For the CIFAR-10 set color histogram is slightly better than SIFT. Interestingly regarding the performance, any hyperparameter have any impact on the performance.

Figure 7 details the training time per fold. On logistic regression SIFT is much faster than Color Histogram.

To gain deeper insights into the misclassifications of the linear models, we generated confusion matrices for the best-performing Logistic Regression configurations found during GridSearch.

Figure 8 depicts the results for CIFAR-10. In contrast to the CV scores, SIFT appears to perform better on CIFAR-10 than the Color Histogram. SIFT is particularly stronger on the higher classes (5–9). However, on

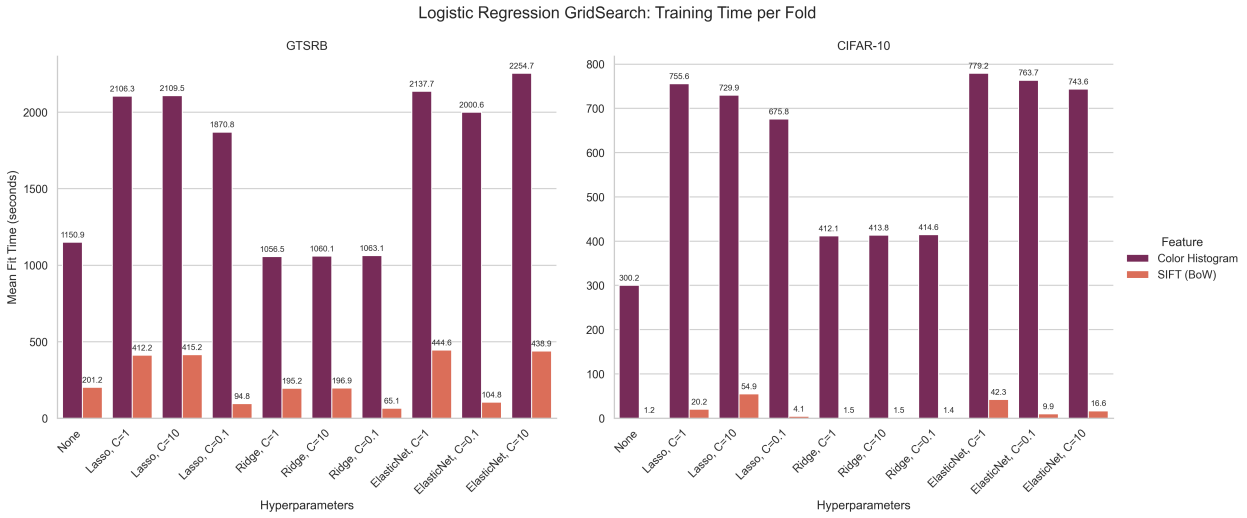


Figure 7: Logistic Regression: Training time per fold.

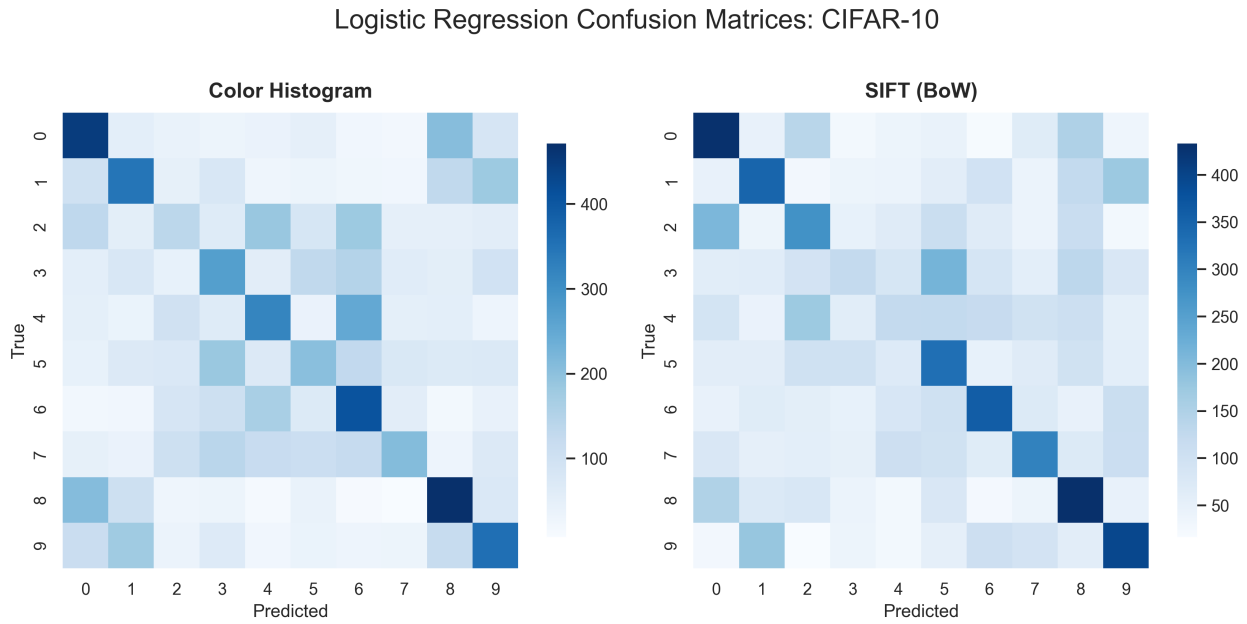


Figure 8: Logistic Regression Confusion Matrices: CIFAR-10.

classes 3 and 4, the Color Histogram is superior, similar to the Random Forest results. Furthermore, on the outside of the diagonal, there are darker points in the Color Histogram matrix compared to SIFT.

Figure 9 displays the confusion matrices for the GTSRB dataset. On this set, SIFT clearly outperforms the Color Histogram. This result aligns perfectly with the accuracy pattern observed in Figure 6. Furthermore, the off-diagonal regions of the SIFT matrix are much cleaner compared to the Color Histogram, indicating significantly fewer misclassifications.

Here, the visualization confirms the performance gap we observed in the accuracy plots. The SIFT matrix (right) exhibits a sharp, well-defined diagonal, indicating that the structural features are well-suited for linear separation.

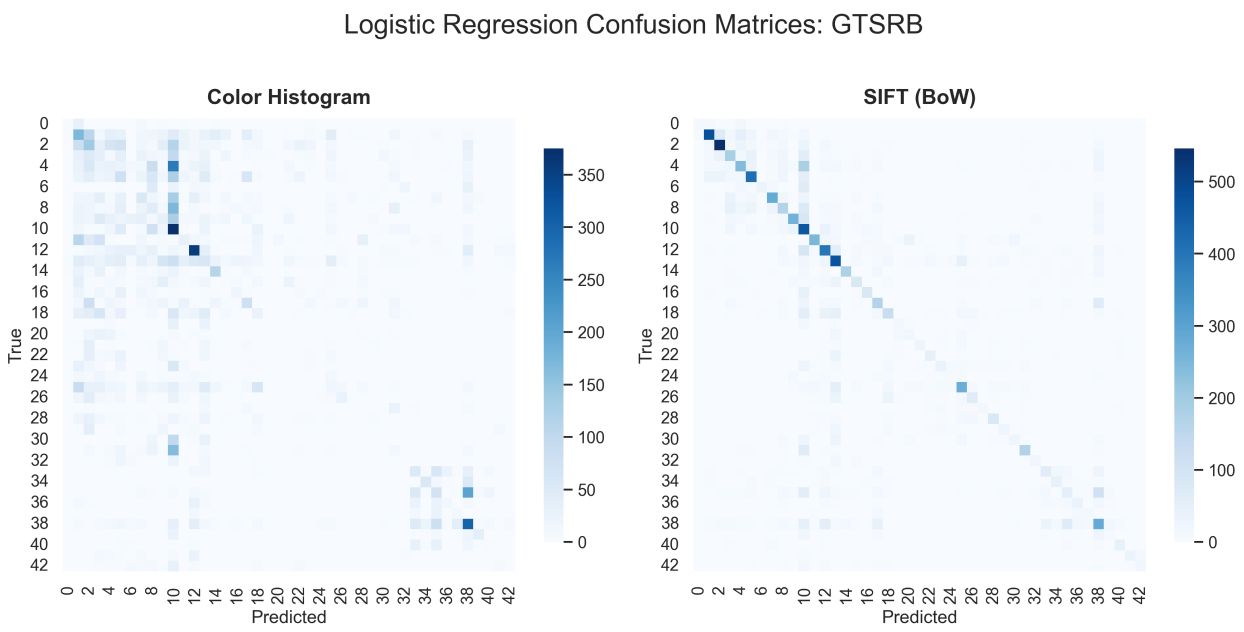


Figure 9: Logistic Regression Confusion Matrices: GTSRB.

Table 2: Logistic Regression Performance. Comparison of GTSRB and CIFAR-10.

| Feature | CV Score | Accuracy | F1 Score | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|
| **GTSRB** | | | | | |
| Color Histogram | 0.3634 ± 0.0000 | 0.1767 | 0.1224 | 2137.74 | 0.0111 |
| SIFT (BoW) | 0.5612 ± 0.0000 | 0.4975 | 0.4268 | 444.64 | 0.0046 |
| **CIFAR-10** | | | | | |
| Color Histogram | 0.3179 ± 0.0000 | 0.3176 | 0.3107 | 413.78 | 0.0092 |
| SIFT (BoW) | 0.3034 ± 0.0000 | 0.3121 | 0.3043 | 1.24 | 0.0024 |

Table 2 provides a detailed summary of the final Logistic Regression performance, highlighting the trade-off between accuracy and computational cost.

On the GTSRB dataset, SIFT is the undisputed winner. Not only does it achieve a significantly higher test accuracy ($\approx 49.8\%$ vs. $\approx 17.7\%$), but it also demonstrates much better generalization. We observe a drastic drop in the Color Histogram's performance, falling from a CV score of 0.36 to a test accuracy of only 0.18. This large discrepancy suggests severe overfitting to the training set or that the color distributions in the test set differ significantly from the training data. SIFT, by contrast, maintains a more stable performance gap between validation and testing.

On CIFAR-10, the accuracy metrics are surprisingly close, with both methods settling around 31%. However, the training time reveals a critical difference. The SIFT (BoW) model trained in a mere **1.24 seconds**, whereas the Color Histogram required over **413 seconds**. This extreme efficiency gain makes SIFT far more attractive for this dataset, as it achieves comparable accuracy at a fraction of the computational cost.

## 3.2 DL approaches

### 3.2.1 LeNet5

Beginning with the CIFAR-10 dataset, we first look at the achieved test accuracy depending on the epoch during the training, shown in Figure 10. The plot clearly shows the possible impact of data augmentation. Without the augmentation, the performance began to decrease very fast. For this reason, in this case we

stopped training with epoch 50. However, in the case where data augmentation has been applied, we see that the test accuracy is, even if slowly, still increasing with every new epoch, converging towards 0.65. In context of the dropout value, the plot shows that a dropout of 0.2 would be more or less the best choice here.

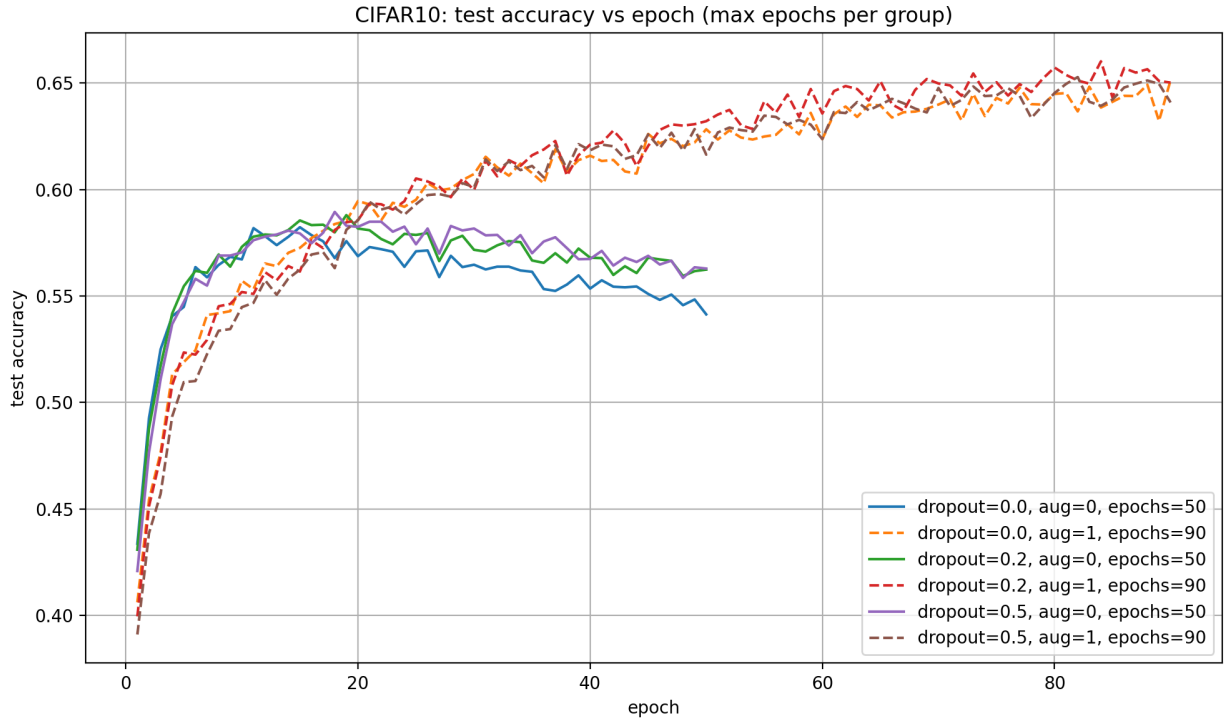


Figure 10: CIFAR10: Test Accuracy vs Epoch

In general, we see that the model is not performing on a great level for this dataset. Intentionally, LeNet5 has been developed to correctly classify data from MNIST (28x28, grayscale), consisting of handwritten digits from 0 to 9. Consequently, this could be the reason behind the not so well performance on this dataset.

The next Figure 11 shows the time needed for training different model configurations. In order to get a meaningful comparison, we looked at the first 50 epochs (= min number of epochs every model was trained for). First, we see that adding augmentation also introduces additional time for training. As every image has to be processed during augmentation, we see that the total train time in this case is approximately three times higher than without augmentation. Besides, we also see that enabling dropout also leads to more training time (adding more computational effort since randomness, masking etc.). However, in this case the difference is negligible. Consequently, we clearly see that augmentation and dropout can improve the performance, but they will also bring in a drawback containing training time.

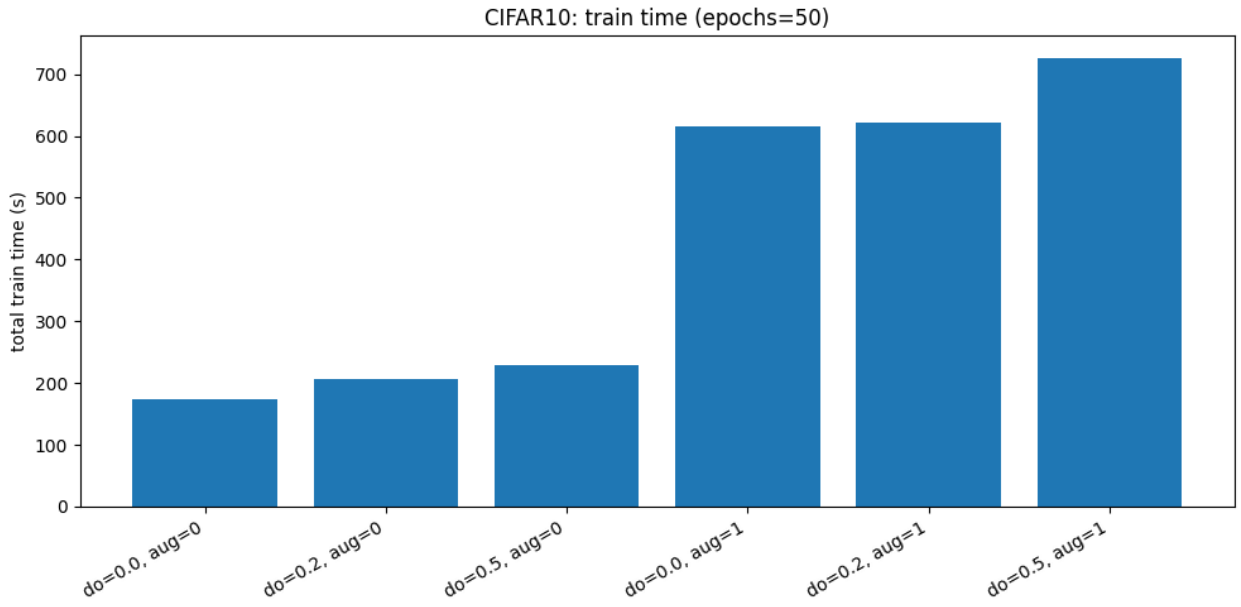


Figure 11: CIFAR10: total train time per configuration

Next, we want to identify which classes are confused with each other by the model and how that changes with the increase of the number of epochs. For this reason, we just look at Figure 12, showing confusion matrices regarding a model with dropout set to zero and augmentation set to 1. The left one is related to the confusion after the first epoch, whereas the right ones shows the matrix after the last epoch, the 90-th epoch in this case. We know that the Test Accuracy after the first epoch is approximately 0.41, whereas being 0.65 for after the 90th one. And we can clearly see that difference in the plots. The confusion matrix got much cleaner on the right side. When we look at the latest matrix, we see that the model still struggles to differ between a dog and a cat, which makes sense. Even after the 90th epoch, the model also cannot distinguish between an automobile and a truck with confidence. The evolution shows that f.e. when in the beginning differing a ship from a truck was challenging, the model more or less learned to distinguish between them. Overall, we clearly see that the performance of the model got a lot better with the high number of epochs.

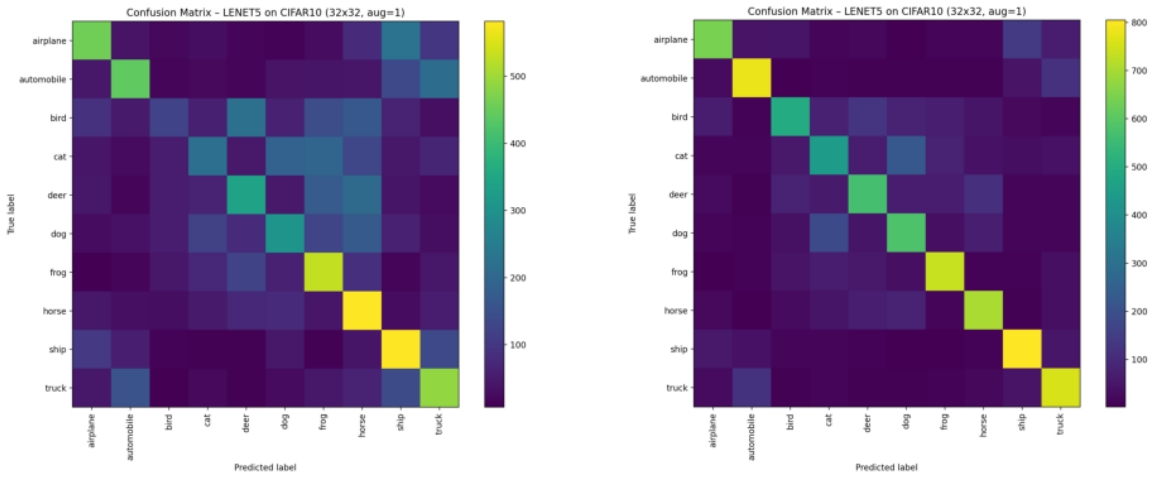


Figure 12: CIFAR10: comparison confusion matrices ep=1 vs. ep=90 (aug=1, do=0)

To demonstrate what a report consists of, Table 3 shows the report for a model with the parameters: epochs=5, dropout=0, augmentation=0. We see different measures are computed, also including the number of instances per class (= support). The last row shows the macro average value. Overall, we do not see really good values, which makes sense since this is still corresponds to the fifth epoch and LeNet5 does not perform well on this dataset anyways. We see that cats are not really recognized. The reason for that is most likely, as mentioned above, that the model cannot distinguish between a cat and a dog. We see that the frogs are being recognized on a certain level. When looking at the macro average value, we see that we have an approximately 50% performance here.

| class | precision | recall | f1-score | support |
|---|---|---|---|---|
| **airplane** | 0.57 | 0.63 | 0.60 | **1000** |
| **automobile** | **0.73** | 0.57 | 0.64 | **1000** |
| **bird** | 0.40 | 0.51 | 0.45 | **1000** |
| **cat** | **0.39** | 0.39 | **0.39** | **1000** |
| **deer** | 0.53 | **0.37** | 0.44 | **1000** |
| **dog** | 0.52 | 0.38 | 0.44 | **1000** |
| **frog** | 0.52 | **0.71** | 0.60 | **1000** |
| **horse** | 0.57 | 0.68 | 0.62 | **1000** |
| **ship** | 0.68 | 0.62 | **0.65** | **1000** |
| **truck** | 0.61 | 0.59 | 0.60 | **1000** |
| **macro avg** | 0.55 | 0.55 | 0.54 | **10000** |

Table 3: CIFAR10: Report after 5 epochs (aug=0, do=0)

To summarize the analysis of different LeNet5 models for this dataset, Figure 13 shows the maximum test accuracies with their corresponding model configurations, epochs and train times. As we see, the model does not perform on a great level for this dataset. Since in the case where no augmentation is applied we just got the model worse with every next epoch, we early stopped here, as already described. So without augmentation, we only get an accuracy of about 59%, but we have a low train time. When applying augmentation, we get best test accuracies (around 66%) at epochs in range [80, 90]. But when we look at the total train time, we see a huge difference. The whole training process gets approximately 6 - 7 times slower.

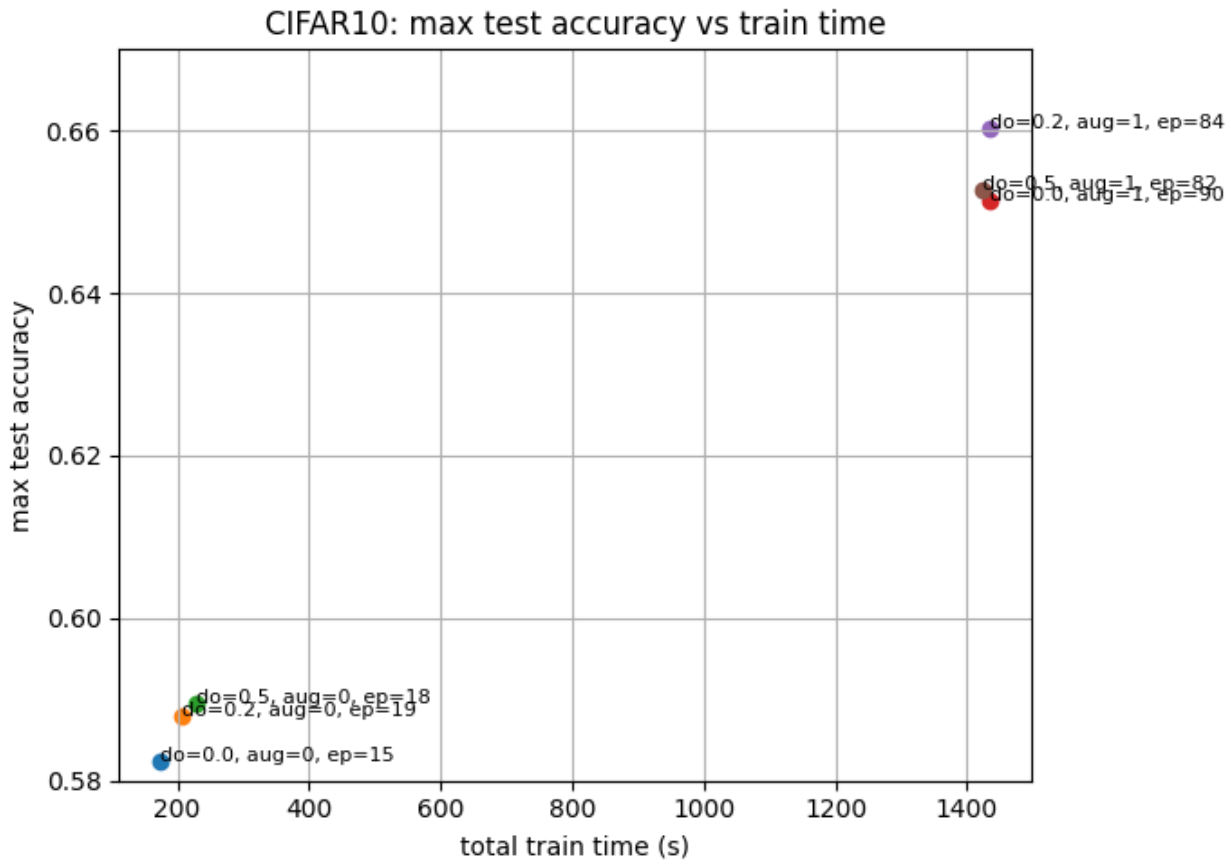


Figure 13: CIFAR10: max test accuracy vs. train time

Figure 14 shows the evolution of the train and evaluation losses and accuracies for the best model. We see no overfitting here, which could be the result of activated dropout. We also see, that the values are always a little bit better for the test / evaluation set. So, predicting the train data was a little bit more difficult than the test data. The reason here could be the data augmentation and the active dropout during training. In general we have an accuracy of about 65 % in the end, which is not the worst result, but we also cannot say that the model is performing on the best level. besides, we had to go through a lot of epochs here, which can also be computationally expensive.

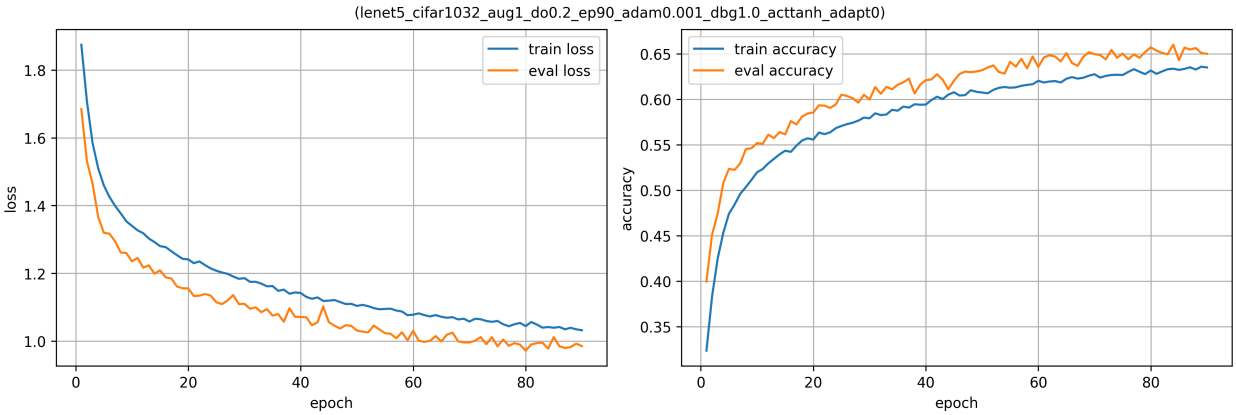


Figure 14: CIFAR-10: Train Evaluation curves for best model (Aug=Yes, Drop=0.2)

Now we continue with the analysis of the LeNet5 performance on the second dataset: GTSRB. Figure 21 again shows the evolution of the test accuracy of several models depending on the epoch number. We see that here, the model performs on a very good level. For this dataset, we show the epoch data until 15, since from this point, the behavior does not change. The test accuracy is already very high just after two epochs, converging towards 93% with every next epoch. We also see that the augmentation has not that big influence as in the other dataset, but in combination with the dropout it still helps increasing the performance. We also have to say that here, a lot of train data exists, so maybe applying augmentation just nearly adds new aspects / patterns to the data.

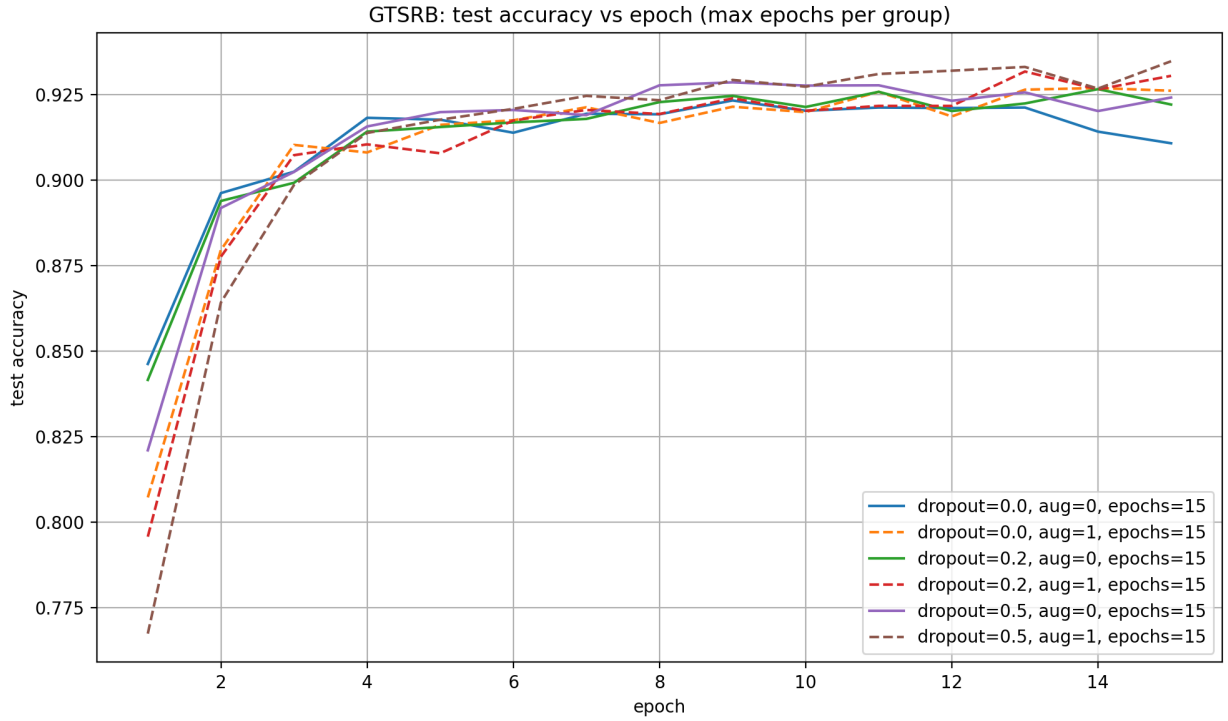


Figure 15: GTSRB: Test Accuracy vs Epoch

Figure 16 again compares the total train time depending on different configurations. We see that we have a high total train time in general, due to the high amount of train data. Additionally, introducing augmentation again triples the amount of time needed for train. The surprising conclusion here is that the total train time decreases when dropout is set to 0.5, with or without augmentation. Maybe, this have been caused by any other background processes when training the model.

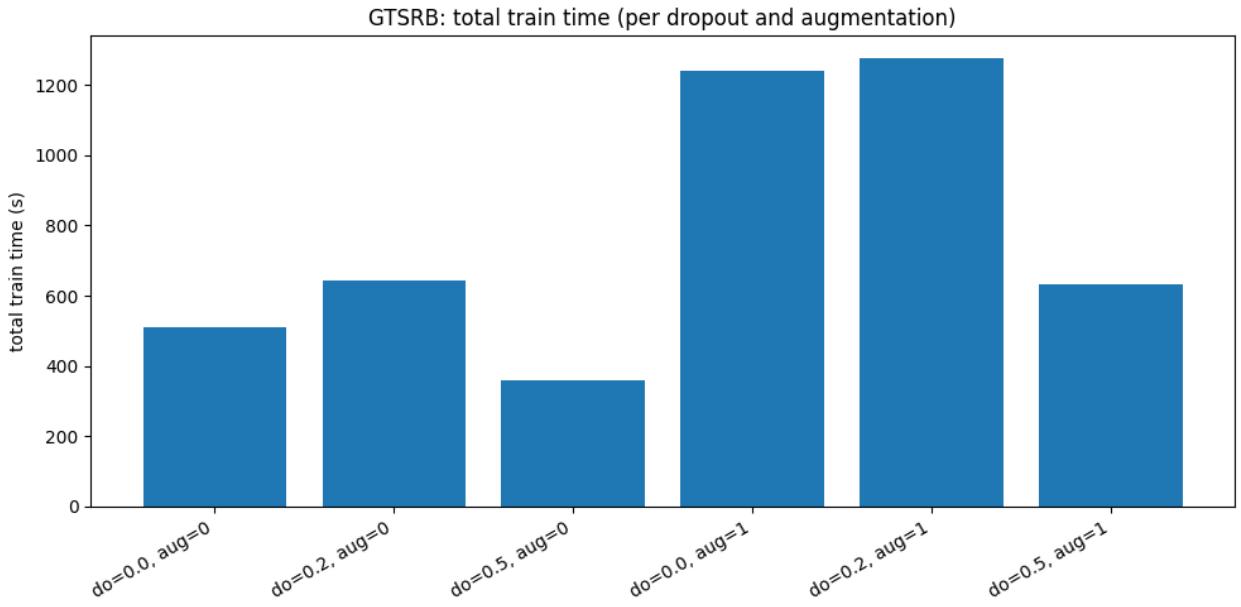


Figure 16: GTSRB: total train time per configuration

Figure 17 compares two confusion matrices of a model configuration, depending on the epoch: Epoch 1 vs. Epoch 15. Dropout is set to zero, whereas augmentation is applied. The accuracies are 80.73% and 92.62%, so the first epoch already has a high accuracy. This can also be seen in the left confusion matrix, which looks clean in most cases. There are just some cases, where the model is confused. When we look at the classes 0 - 8, we see there are some confusions. And this makes sense, when we look at what these classes represent:

- class 0: Speed limit (20km/h)
- class 1: Speed limit (30km/h)
- class 2: Speed limit (50km/h)
- class 3: Speed limit (60km/h)
- class 4: Speed limit (70km/h)
- class 5: Speed limit (80km/h)
- class 6: End of speed limit (80km/h)
- class 7: Speed limit (100km/h)
- class 8: Speed limit (120km/h)

We also see that the following classes are confused with especially class 31 (Wild animals crossing):

- class 19: Dangerous curve to the left
- class 21: Double curve
- class 23: Slippery road
- class 24: Road narrows on the right
- etc.

So, when looking at the signs the classes are representing, it somehow makes sense that the model is confused here, since the shapes, colors etc. are quite similar in these cases. When we then look at the right confusion matrix, we see that most of the confusions have been fixed by training.

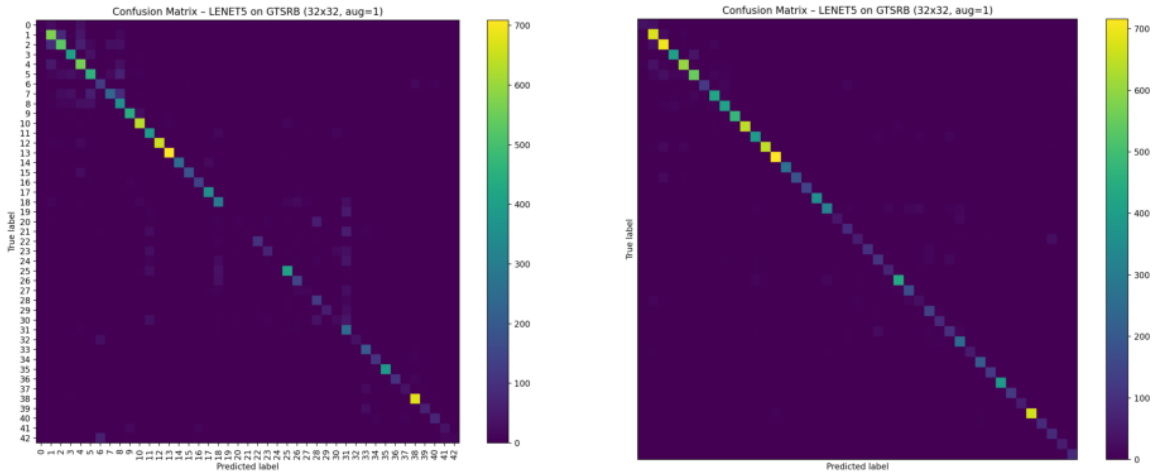


Figure 17: GTSRB: comparison confusion matrices ep=1 vs. ep=15 (aug=1, do=0)

Figure 18 relates the maximum test accuracy achieved by a model to its corresponding total train time, the model parameters, and the epoch where this performance have been achieved. First, we clearly see that enabling augmentation leads to better performance, but increases the train time as well. So, which model to choose highly depends on the specific task. F.e. if someone cannot afford much time consuming, then one will rather take the model without augmentation and dropout. In conclusion, in any configuration case, the LeNet5 performs very well on the GTSRB dataset.

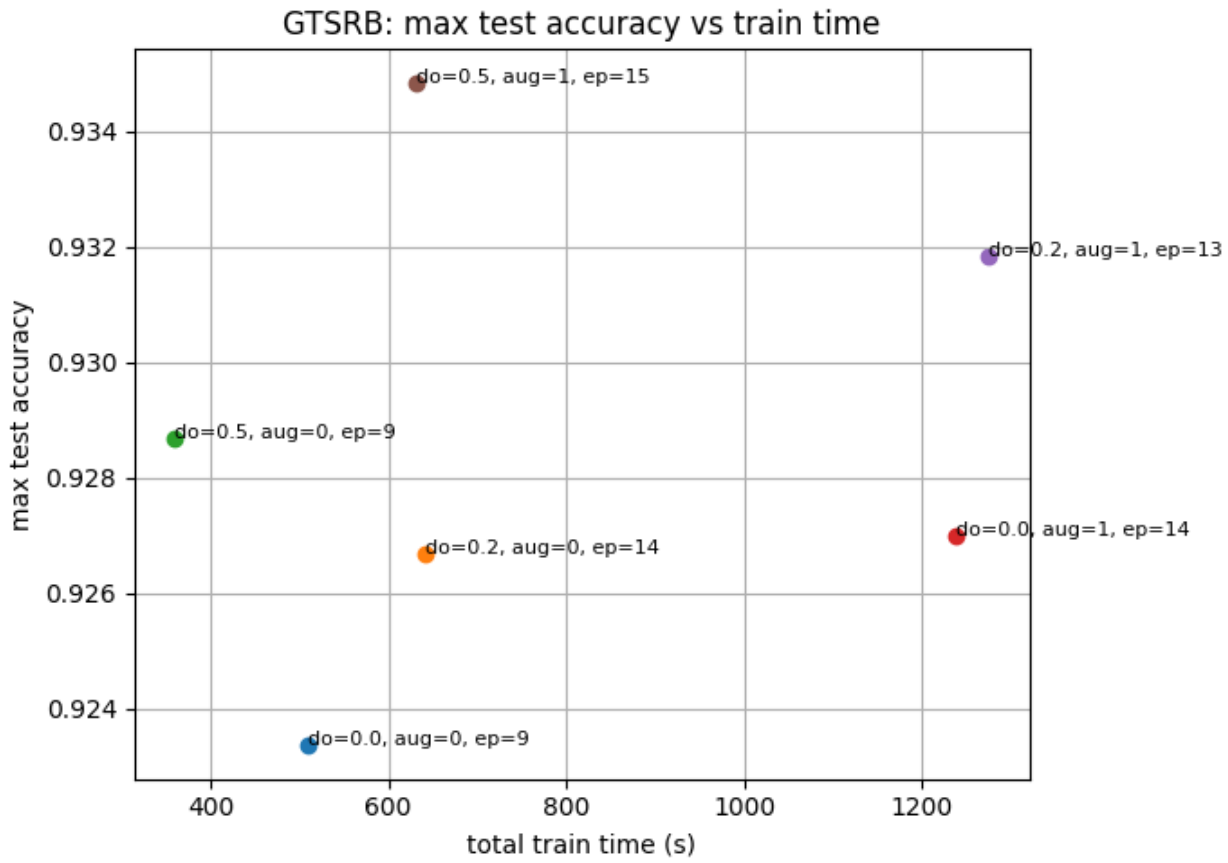


Figure 18: GTSRB: max test accuracy vs. train time

Figure 19 shows again the performance of the best model during the train and evaluation process of each epoch. We see we have a really fast convergence here. After 4 - 6 epochs, we reach a more or less stable evaluation accuracy here with 90 %. So the model performs really well for this big dataset. We also see a minimal overfitting, which begins with the fifth epoch. So, at this point, the model still gets better when predicting train data, but cannot generalize in a better way anymore.

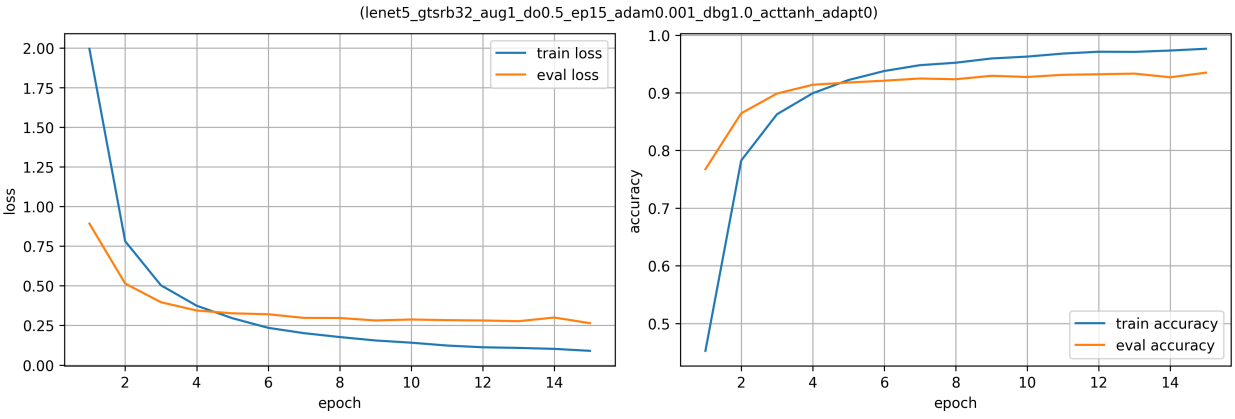


Figure 19: GTSRB: Train & Evaluation curves for best model (Aug=Yes, Drop=0.5)

### 3.2.2 ResNet18

We trained ResNet-18 on the GTSRB dataset with 18 different configurations, varying pre-training (Pre), layer freezing (Frz), data augmentation (Aug), and dropout rates (Drop). Each model was trained for 32 epochs to see which setup works best for traffic sign classification.

Looking at Figure 20, the most obvious pattern jumps out immediately: freezing the pre-trained layers completely kills the model performance. All frozen configurations (shown in red) get stuck at around 35-37% accuracy, while non-frozen models (in blue) reach 93-96%. That's a massive 60 percentage point difference.

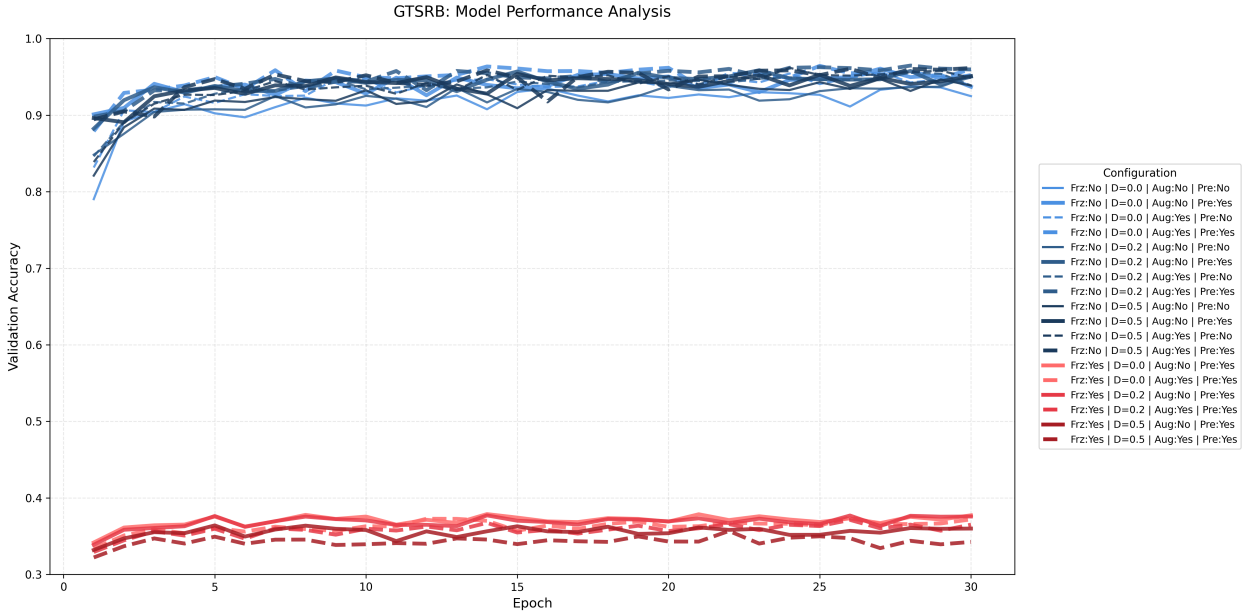


Figure 20: GTSRB: Test Accuracy vs Epoch

The problem with freezing becomes clear when we consider what traffic signs actually look like. They have very specific characteristics: geometric shapes, distinctive color patterns, and symbolic content that's quite different from natural images in ImageNet. When we freeze the layers, we're forcing the model to use ImageNet features which are not suitable for our data. The network simply can't adapt these features to what it actually needs to recognize.

Once we allow all layers to train (Frz=No), performance shoots up to the 93-96% range across all configurations which we can see Figure **??**. This shows that the model needs the freedom to adjust those pre-trained features to work with traffic signs.

Figure 22 breaks down the impact of individual hyperparameters for all non-frozen configurations.

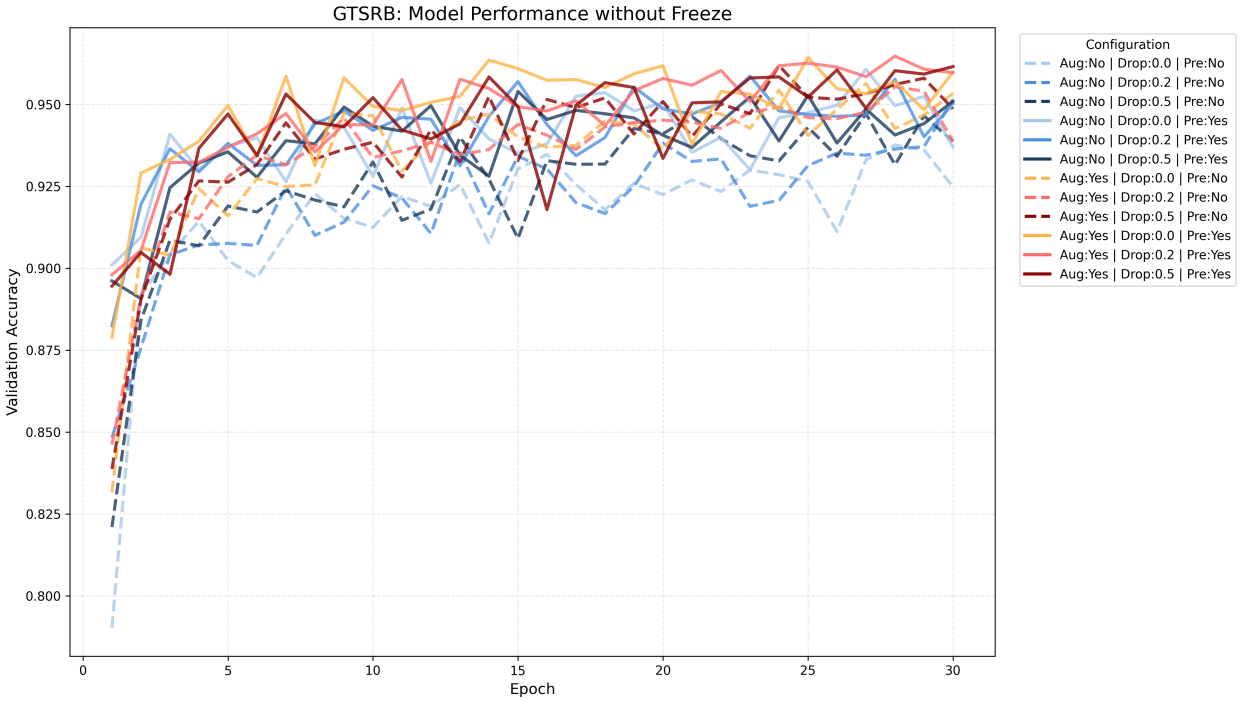


Figure 21: GTSRB: Test Accuracy vs Epoch (without freeze)

Data augmentation provides a clear benefit, pushing the median accuracy from around 95%. This makes sense for traffic signs, which can appear under various lighting conditions and angles in real-world scenarios. Dropout rate shows minimal impact. But the models without dropouts tend to perform slightly worse. Pre-trained weights offer a 1-2 percentage point improvement.

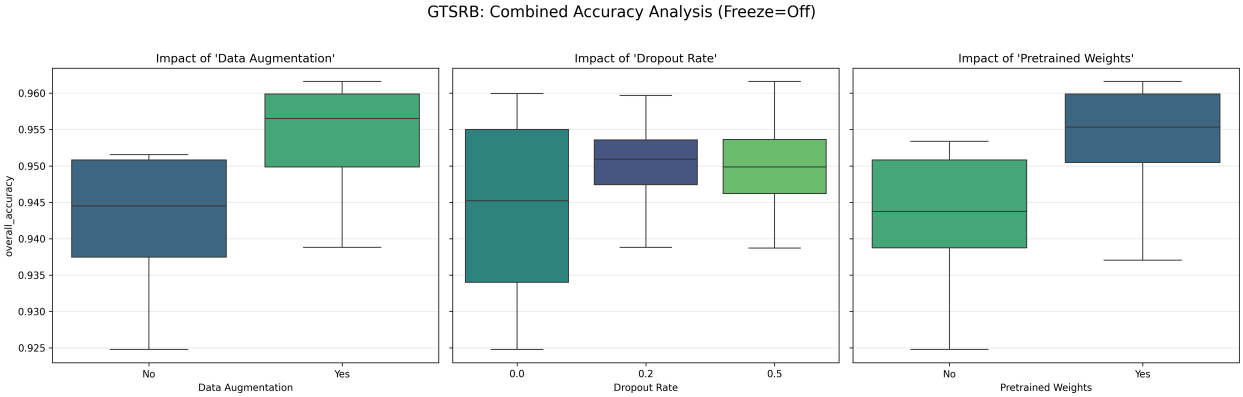


Figure 22: GTSRB: Test Accuracy vs Epoch (without freeze)

Figure 23 shows the training time for each configuration. Frozen models are significantly faster, taking only 3-6 minutes compared to 8-14.5 minutes for non-frozen models. This makes sense since freezing means fewer parameters need gradient computation and updates. Configurations with augmentation also tend to take longer since each training batch requires additional image transformations. However, as we'll see in the trade-off analysis, this extra training time is clearly justified by the massive performance improvement.

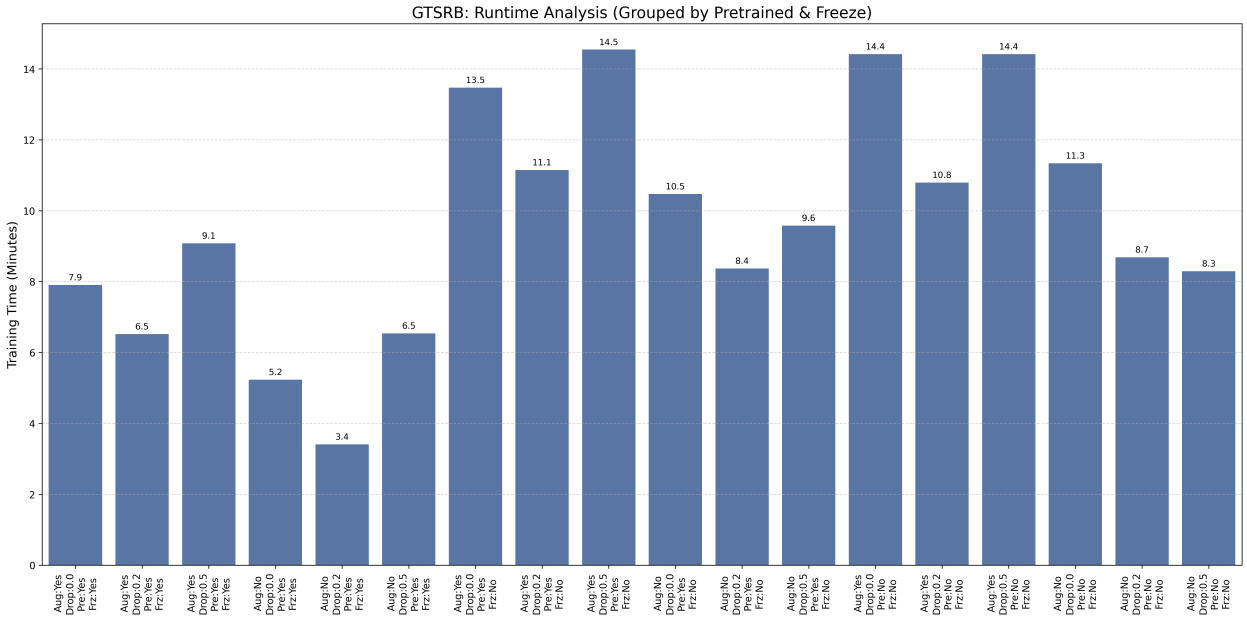


Figure 23: GTSRB: total train time per configuration

Figure 24 shows the runtime vs. accuracy for all non-frozen configurations. All models cluster tightly in the 93-96% accuracy range, with training times varying between 8 and 15 minutes.

The key observation is that runtime doesn't correlate with better performance within this group. A model that takes 14 minutes doesn't perform noticeably better than one that takes 8 minutes - they all end up around 93-96%. The runtime differences are mainly driven by computational factors like augmentation and whether pre-trained weights are used, not by improved accuracy.

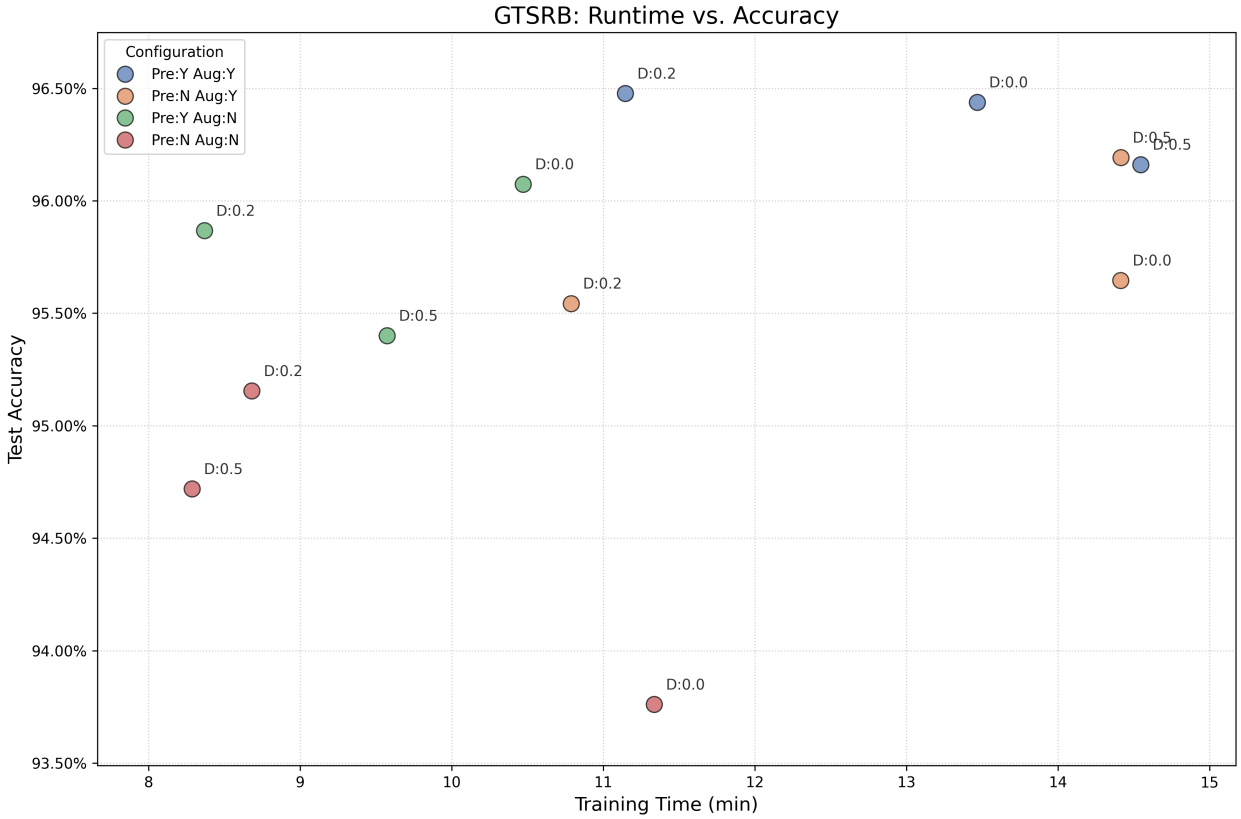


Figure 24: GTSRB: max test accuracy vs. train time

Our best configuration (Pre=Yes, Frz=No, Aug=Yes, Drop=0.0) achieves 96.2% overall accuracy with weighted averages of 96.3% precision, 96.2% recall, and 96.1% F1-score. To better understand the model's strengths and weaknesses, we examine the top and bottom performing classes in Table 4.

| Rank | Class | Sign Type | Prec. | Rec. | F1 | Test Samples |
|---|---|---|---|---|---|---|
| **Top 5 Classes** | | | | | | |
| 1 | 20 | Dangerous curve to the right | 1.000 | 1.000 | 1.000 | 90 |
| 2 | 14 | Stop | 0.996 | 1.000 | 0.998 | 270 |
| 3 | 10 | No passing for vehicles over 3.5 metric tons | 1.000 | 0.992 | 0.996 | 660 |
| 4 | 17 | No entry | 1.000 | 0.992 | 0.996 | 360 |
| 5 | 16 | Vehicles over 3.5 metric tons prohibited | 0.987 | 1.000 | 0.993 | 150 |
| **Bottom 5 Classes** | | | | | | |
| 39 | 18 | General caution | 0.946 | 0.897 | 0.921 | 390 |
| 40 | 37 | Go straight or left | 0.843 | 0.983 | 0.908 | 60 |
| 41 | 40 | Roundabout mandatory | 0.792 | 0.933 | 0.857 | 90 |
| 42 | 30 | Beware of ice/snow | 0.803 | 0.840 | 0.821 | 150 |
| 43 | 27 | Pedestrians | 0.795 | 0.517 | 0.626 | 60 |

Table 4: GTSRB: Top 5 and Bottom 5 Classes by F1-Score

The top-performing classes achieve near-perfect recognition. "Dangerous curve to the right" (Class 20) reaches 100% across all metrics, likely due to its highly distinctive curved arrow symbol. "Stop" signs (Class 14) also excel with 99.8% F1-score - their unique octagonal shape and red color make them easily recognizable. Similarly, "No entry" (Class 17) and "No passing for vehicles over 3.5 metric tons" (Class 10) achieve over 99

The bottom classes reveal interesting challenges. "Pedestrians" (Class 27) is the clear outlier with only 62.6% F1-score. The low recall (51.7%) combined with decent precision (79.5%) means the model often fails to detect this sign. With only 60 training samples and a relatively complex pictogram (human figures), this class lacks sufficient data to learn robust features.

"Beware of ice/snow" (Class 30, 82.1% F1) shows balanced but mediocre performance, possibly because the snowflake symbol is visually complex. "Roundabout mandatory" (Class 40, 85.7% F1) has an interesting pattern: high recall (93.3%) but low precision (79.2%). This could suggests the model frequently misidentifies other circular signs with arrows as roundabout signs, indicating visual similarity with directional signs.

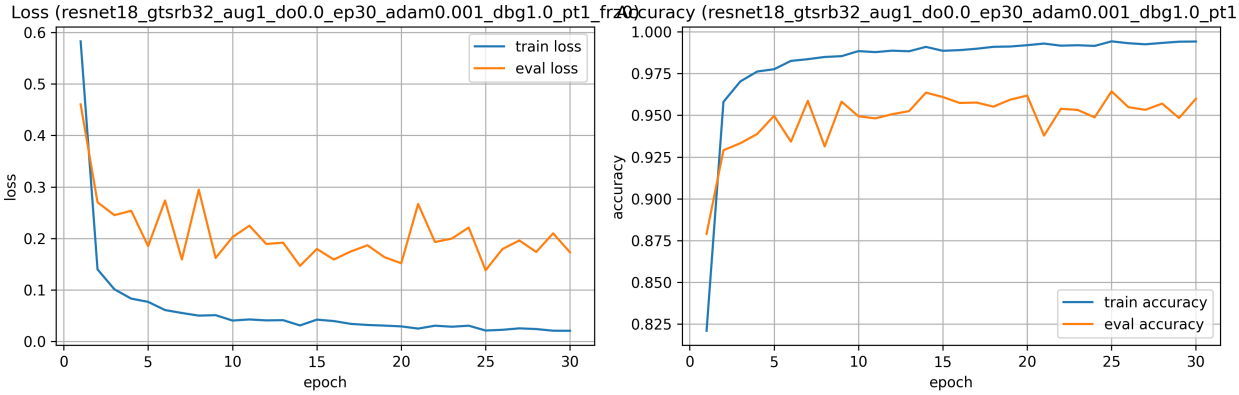


Figure 25: GTSRB: Train Evaluation curves for best model (Pre=Yes, Frz=No, Aug=Yes, Drop=0.0)

The training curves in Figure 25 show rapid convergence after 10 epochs, reaching around 95% evaluation accuracy. Validation accuracy, on the other hand, stagnates after 7 epochs at around 95-96% with notable oscillations.

The gap between training and validation accuracy is larger here than in CIFAR-10, reaching about 4-5 percentage points. However, the evaluation loss remains stable around 0.15-0.20 without increasing, while training loss continuously drops toward nearly zero. This behavior shows why dropout=0.0 works well here. Data augmentation provides sufficient regularization to prevent harmful overfitting.

We applied the same experimental protocol to CIFAR-10, training ResNet-18 with 18 different configurations across the same hyperparameters: pre-training (Pre), freezing (Frz), augmentation (Aug), and dropout (Drop). Each model was trained for 30 epochs.

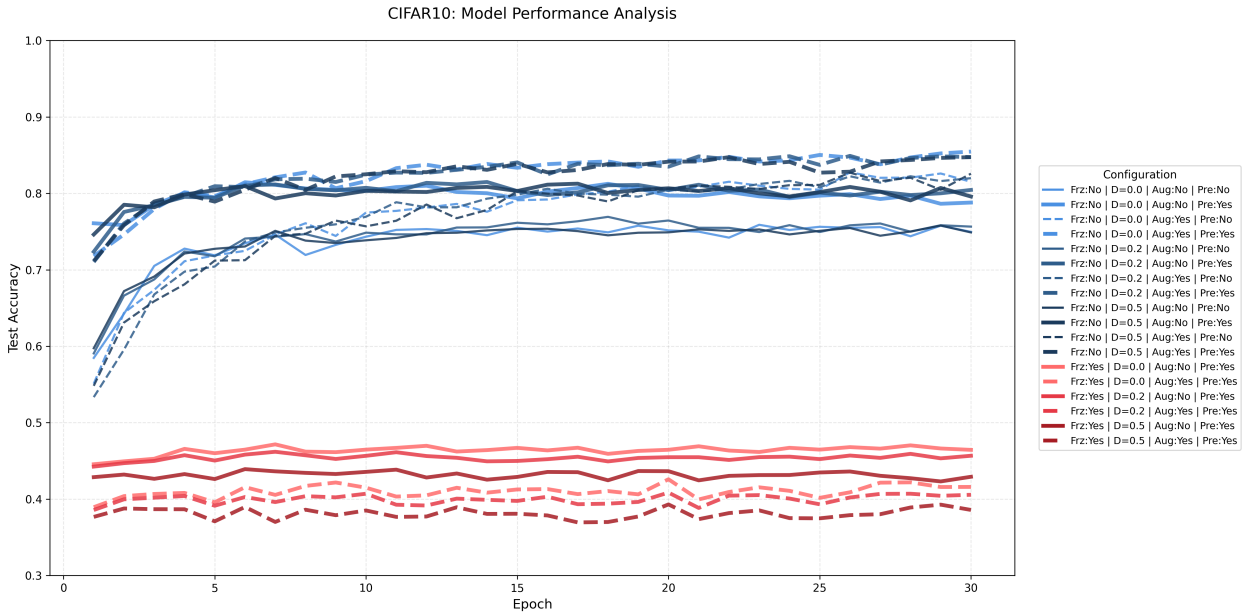


Figure 26: CIFAR-10: Test Accuracy vs Epoch

Figure 26 reveals a familiar pattern as GTSRB: frozen models (red) fail completely at 37-47% accuracy, while non-frozen models (blue) reach 75-85%. That's a 40 percentage point gap.

However, unlike GTSRB, the explanation here is different. ImageNet actually contains very similar content to CIFAR-10 - both include animals, vehicles, everyday objects, and household items. We think CIFAR-10 uses tiny 32x32 pixel images, while ResNet-18 was pre-trained on ImageNet's 224x224 images. At such low resolution, there's simply not enough visual information for the frozen ImageNet features to work with.

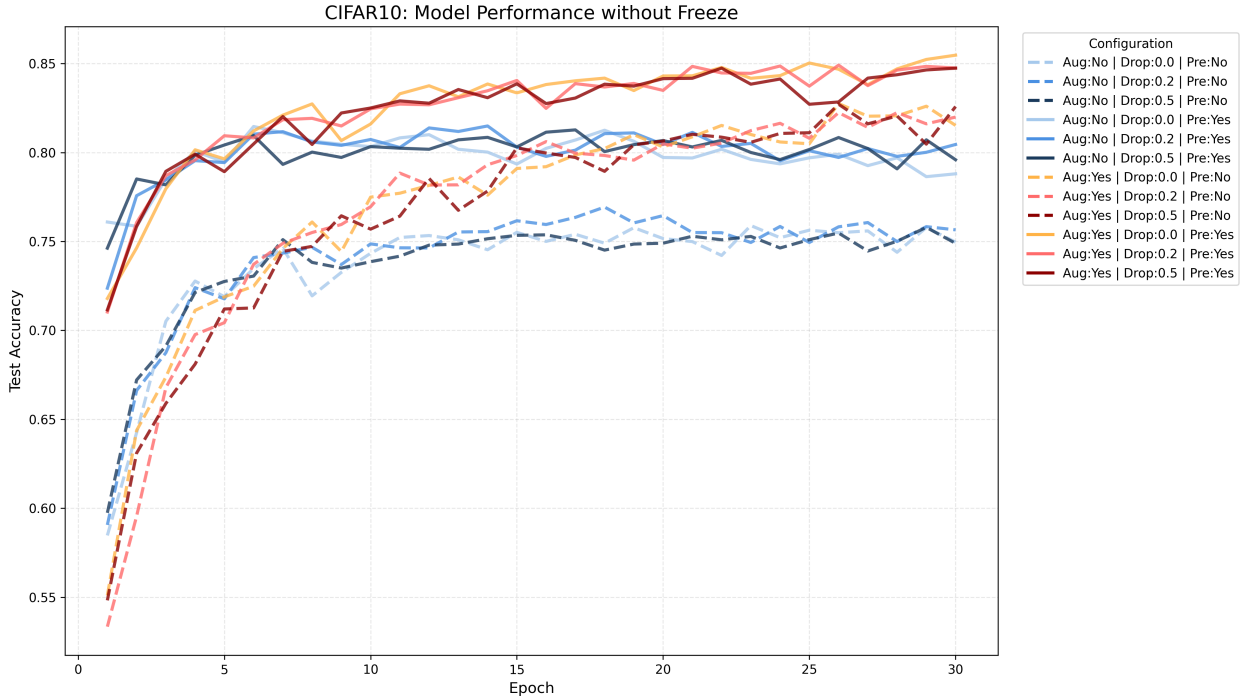


Figure 27: GTSRB: Test Accuracy vs Epoch (without freeze)

For further analysis, we refer to Figure 27, which illustrates the performance of all models without freezing. In this visualization, a clearer separation emerges between pretrained and non-pretrained models, as well as between those with and without data augmentation. The pretrained models perform significantly better, which can be attributed to the fact that ImageNet shares more similarities with CIFAR-10 than it does with GTSRB.

While the performance of non-augmented, pretrained models remains mostly stagnant after 15 epochs, the augmented, pretrained models reach their peak around the 20-epoch. The non-pretrained, augmented models show the most significant improvement as epochs increase and might even continue to improve beyond 30 epochs. After 20 epochs, these models eventually surpass the performance of the pretrained, non-augmented models.

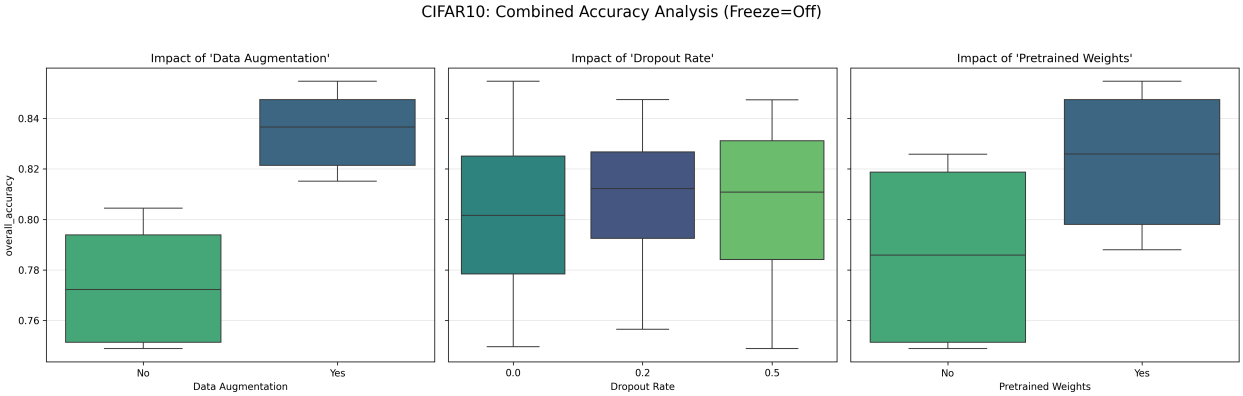


Figure 28: GTSRB: Test Accuracy vs Epoch (without freeze)

Figure 28 breaks down the impact of individual hyperparameters for all non-frozen configurations. Data augmentation exerts the strongest influence, boosting the median accuracy from approximately 77% to 83.5%. This substantial improvement underscores the necessity of augmentation for effectively learning features from the limited visual information in small 32x32 images. Similarly, pre-trained weights provide a distinct advantage, raising the median accuracy from 78% to 82.5%. This confirms that the semantic features learned from ImageNet transfer effectively to CIFAR-10. In contrast, the dropout rate shows almost no impact on the results. All settings (0.0, 0.2, 0.5) produce nearly identical accuracy distributions.

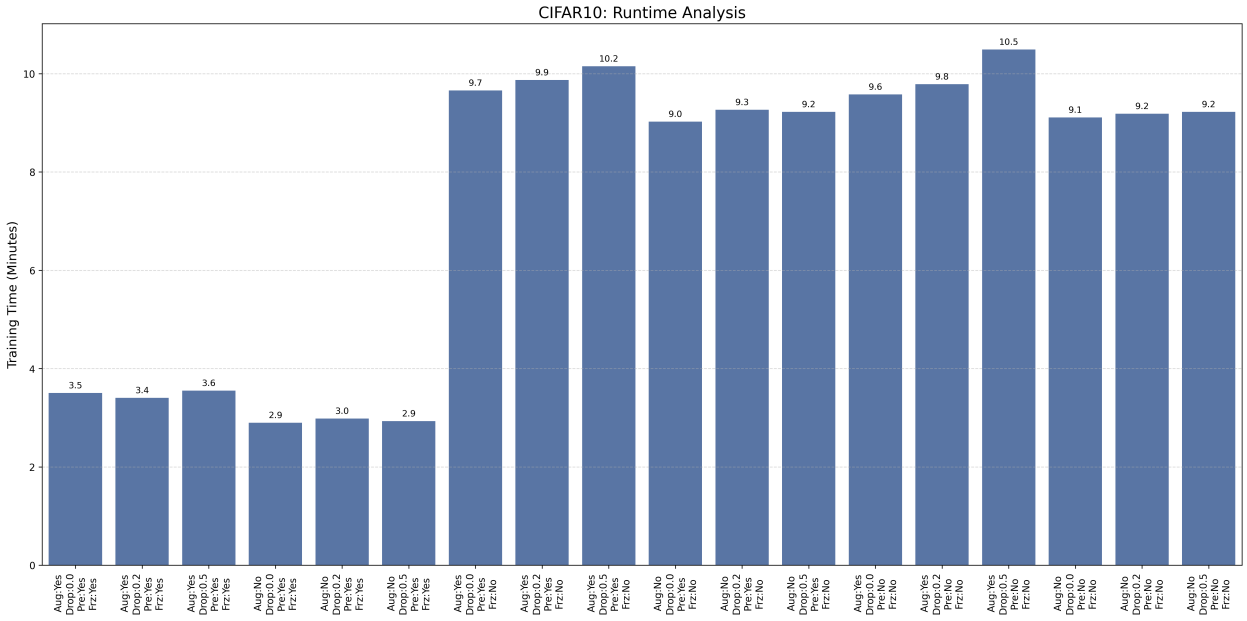


Figure 29: CIFAR-10: Runtime Analysis

The training times follow a similar pattern to GTSRB. As shown in Figure 29, frozen models are much faster, taking only 2.9-3.6 minutes, while non-frozen models require 9.0-10.5 minutes. This makes sense since freezing means fewer parameters need gradient updates.

Within the non-frozen models, configurations with augmentation take slightly longer (9.7-10.5 minutes) compared to those without augmentation (9.0-9.6 minutes). This is expected since augmentation requires additional image transformations for each training batch.

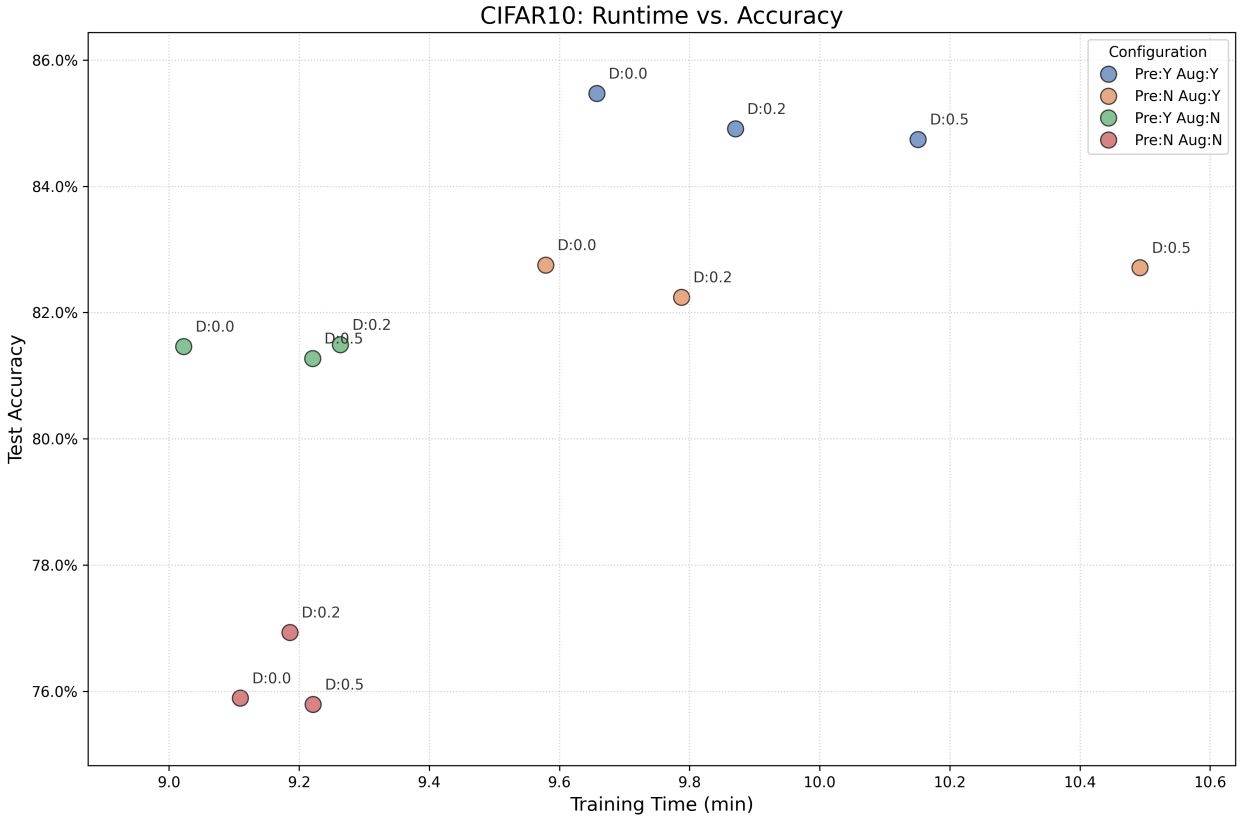


Figure 30: CIFAR-10: Trade-off - Runtime vs Accuracy (Without Freeze)

Looking at the runtime-accuracy trade-off for non-frozen models in Figure 30, we can see clear clusters. The non-augmented models are weaker but faster. Pre-trained models perform much better but do not require extra training time. In contrast, in almost every group, increasing the dropout rate makes the models take longer to train but does not improve performance.

| class | precision | recall | f1-score | support |
|---|---|---|---|---|
| **airplane** | 0.780 | 0.931 | 0.849 | **1000** |
| **automobile** | 0.915 | 0.936 | 0.925 | **1000** |
| **bird** | 0.861 | 0.822 | 0.841 | **1000** |
| **cat** | 0.706 | 0.719 | 0.713 | **1000** |
| **deer** | 0.853 | 0.871 | 0.862 | **1000** |
| **dog** | 0.806 | 0.731 | 0.767 | **1000** |
| **frog** | 0.868 | 0.912 | 0.889 | **1000** |
| **horse** | 0.919 | 0.856 | 0.887 | **1000** |
| **ship** | 0.934 | 0.902 | 0.918 | **1000** |
| **truck** | 0.927 | 0.867 | 0.896 | **1000** |
| **macro avg** | 0.857 | 0.855 | 0.855 | **10000** |
| **weighted avg** | 0.857 | 0.855 | 0.855 | **10000** |

Table 5: CIFAR-10: Best Model Report (Pre=Yes, Frz=No, Aug=Yes, Drop=0.0)

Our best performing configuration uses Pre=Yes, Frz=No, Aug=Yes, and Drop=0.0, achieving an overall accuracy of 85.5%. As shown in Table 5, performance varies considerably across classes, ranging from 71.3% to 92.5% F1-score.

Cat and dog are the weakest performers at 71.3% and 76.7% F1-score respectively. At 32x32 resolution, furry four-legged animals are difficult to distinguish. Airplane shows an unusual pattern with high recall (93.1%) but much lower precision (78.0%). This suggests the model often misclassifies other objects as airplanes, possibly confusing them with birds at low resolution. In contrast, vehicles (automobile, ship, truck) perform best with F1-scores above 89%, likely because their rigid shapes remain distinctive even in small images.

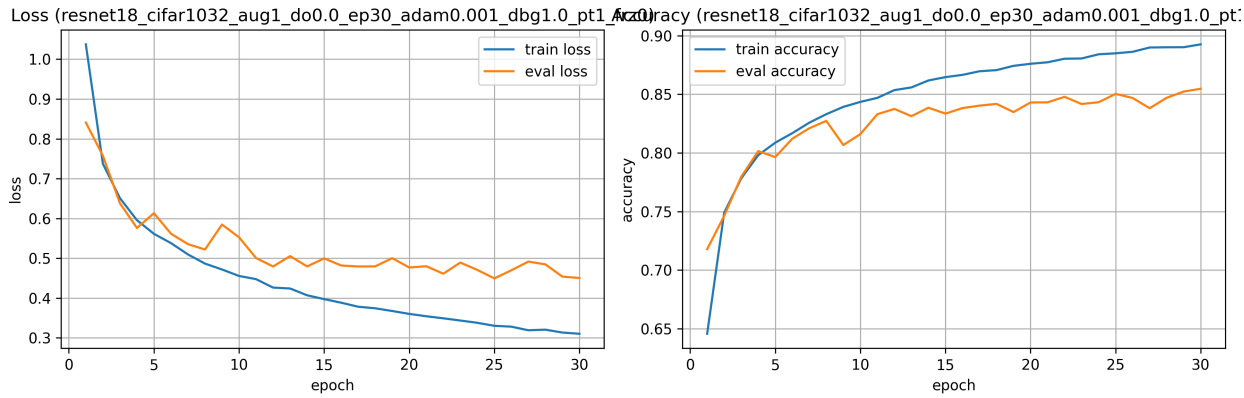


Figure 31: CIFAR-10: Train Evaluation curves for best model (Pre=Yes, Frz=No, Aug=Yes, Drop=0.0)

The training curves in Figure 31 show that performance rises and is still increasing at 30 epochs. Validation accuracy, on the other hand, stagnates after 20 epochs. So with higher epochs, there is a possibility for overfitting.

Crucially, the evaluation loss remains stable around 0.45–0.50 without decreasing after 12 epochs. The training loss is constantly sinking, likely even after 30 epochs. This controlled behavior shows why dropout is not necessary here.

# 4 Conclusion: ML vs. DL

Now, we will compare all the different algorithms and models applied to each dataset. For every DL model, we will print a table containing the top 3 models based on the test accuracy. For the ML algorithms, we print the best model per feature extraction method.

### 4.0.1 CIFAR-10

Tables 6, 7, 8 and 9 show the results for each ML and DL approach. We directly see that the ML algorithms had the worst performance here. For Logistic Regression, the feature extraction method nearly changed anything in terms of accuracy, whereas for the Random Forest method, the feature extraction method had an impact on the accuracy. Among the ML approaches, Random Forest is the clear winner.

When we look at the DL approaches, we see that they are much better than the ML approaches. First, the LeNet5 model performs with 66 % significantly better than the ML approaches. However, the ResNet18 model is again significantly better than LeNet5, although the number of epochs here is much lower. When we look at train time, we also see that ResNet18 did need muhc less time than LeNet5. We must not forget that the effect of transfer learning / pretraining can be seen here in the ResNet18 case. We also see that in any top DL approach, data augmentation has been applied, which shows how helpful this method can be. When we consider the LeNet5 models, we see that the Dropout rate does not have a huge impact on the results. Besides, in the ResNet18 case, we even have a best model which does not use dropout during training at all.

Overall, the ResNet18 model with about 85 % test accuracy can be considered as the clear winner here.

| Feature | Penalty | C | Test Acc (%) | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|
| Color Hist | Ridge | 10 | 31.76 | 413.78 | 0.0092 |
| SIFT BoVW | None | - | 31.21 | 1.24 | 0.0024 |

Table 6: CIFAR-10: Best Logistic Regression results per Feature Extraction Method

| Feature | Max Depth | Trees | Test Acc (%) | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|
| Color Hist | 15 | 90 | 41.96 | 4.27 | 0.0408 |
| SIFT BoVW | 15 | 90 | 28.66 | 1.87 | 0.0248 |

Table 7: CIFAR-10: Best Random Forest results per Feature Extraction Method

| Drop | Aug | Test Acc (%) | Epochs | Train time (s) | Test time (s) |
|---|---|---|---|---|---|
| 0.2 | 1 | 66.02 | 84 | 1343.97 | 77.65 |
| 0.5 | 1 | 65.28 | 82 | 1295.78 | 77.67 |
| 0.0 | 1 | 65.14 | 90 | 1433.94 | 82.75 |

Table 8: CIFAR10: Top 3 LeNet5 models

| Drop | Aug | Pre | Frz | Test Acc (%) | Epochs | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|---|---|
| 0.0 | 1 | 1 | 0 | 85.47 | 30 | 579.46 | 72.06 |
| 0.2 | 1 | 1 | 0 | 84.91 | 26 | 515.94 | 64.15 |
| 0.5 | 1 | 1 | 0 | 84.74 | 22 | 447.67 | 80.88 |

Table 9: CIFAR-10: Top 3 ResNet18 models (Hyperparameters: Drop=Dropout, Aug=Augmentation, Pre=Pretrained, Frz=Freeze)

#### 4.0.2 GTSRB

Tables 10, 11, 12 and 13 show the results for each ML and DL approach on GTSRB. Compared to CIFAR-10, the ML baselines behave differently: the feature extraction method has a substantial impact. For Logistic Regression, Color Hist performs very poorly, whereas SIFT BoVW reaches almost 50 %. A similar pattern is visible for Random Forest, where Color Hist achieves 23.13 % but SIFT BoVW reaches 47.21 %, which is twice as much. So, here the choice of the feature extraction method is really important for the performance. Among these models, Logistic Regression with SIFT BOVW is the best model.

For the DL approaches, we again see a huge improvement over ML approaches. ML. LeNet5 already achieves strong results around 93 %, which is much better then the ML approaches. ResNet18 provides an additional improvement to about 96.5 % test accuracy, which makes it the best one. Here, the benefit of transfer learning / pretraining seems to be smaller here, because the top pretrained ResNet18 models are only slightly better than the best non-pretrained run. We also see that Data Augmentation is still important, but does not have that much of a relevance as in the CIFAR-10 case. We see that for the LeNEt 5 models, we also have one model without augmentation, still performing well. Regarding dropout, the results again indicate that the inluence of on performance is also limited here.

Overall, ResNet18 with about 96 % test accuracy can be considered the clear winner on GTSRB, while LeNet5 already offers a strong and more lightweight alternative with circa 93 % accuracy.

| Feature | Penalty | C | L1 Ratio | Test Acc (%) | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|---|
| Color Hist | ElasticNet | 1 | 0.5 | 17.67 | 2137.74 | 0.0111 |
| SIFT BoVW | ElasticNet | 1 | 0.5 | 49.75 | 444.64 | 0.0046 |

Table 10: GTSRB: Best Logistic Regression results per Feature Extraction Method

| Feature | Max Depth | Trees | Test Acc (%) | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|
| Color Hist | 15 | 90 | 23.13 | 3.79 | 0.0711 |
| SIFT BoVW | 15 | 90 | 47.21 | 1.87 | 0.0485 |

Table 11: GTSRB: Best Random Forest results per Feature Extraction Method

| Drop | Aug | Test Acc (%) | Epochs | Train time (s) | Test time (s) |
|---|---|---|---|---|---|
| 0.5 | 1 | 93.48 | 15 | 631.98 | 84.83 |
| 0.2 | 1 | 93.18 | 13 | 1104.16 | 144.65 |
| 0.5 | 0 | 92.87 | 9 | 254.71 | 64.33 |

Table 12: GTSRB: Top 3 LeNet5 models

| Drop | Aug | Pre | Frz | Test Acc (%) | Epochs | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|---|---|
| 0.2 | 1 | 1 | 0 | 96.48 | 28 | 622.51 | 202.03 |
| 0.0 | 1 | 1 | 0 | 96.44 | 25 | 666.10 | 206.22 |
| 0.5 | 1 | 0 | 0 | 96.19 | 24 | 689.14 | 212.57 |

Table 13: GTSRB: Top 3 ResNet18 models (Hyperparameters: Drop=Dropout, Aug=Augmentation, Pre=Pretrained, Frz=Freeze)