CS6482 Deep Reinforcement Learning

Assignment 1: Sem2 AY 23/24 – Convolutional Neural Networks (CNNs)

RESNET18 WITH SE BLOCK ARCHITECTURE WITH CIFAR 10 DATASET

SUBMITTED BY: 1. SAJIN MOHAMED PALLIKKATHODI ERATHALI(23037601)

2. AKHIL RAGHU NATH (23048417)

Table of Contents

The library we used to complete the assignment is Tensor Flow as we had worked with PyTorch in the previous semester, we wanted to explore Tensor Flow. This assignment help us investigate CNNs and various architectures before committing ourselves to ResNet(which was the state-of the art CNN until Yolo came along). We chose ResNet18 over a complex ResNet50 or ResNet150, for a simple enough architecture which would enable us to explore the impact of variation of hyper parameters more prominently.

1. Data Set

- The CIFAR-10 dataset was created by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It consists of 60000 images of 32 x 32 pixels, where we have 10 classes(airplane, automobile, bird etc.) each having 6000 sample images per class.

1.1. Visualization of Dataset

- A sample from the dataset is illustrated in Figure 1. We started with a simple dataset for analysis for a simple ResNet Architecture(ResNet-18).

Figure 1. CIFAR-10 Dataset.

- The CIFAR-10 dataset gave promising result with ResNet-18 architecture. Since this was the dataset. We wanted to use ImageNet Dataset, however due to the size of the dataset being so large in the range of 100 GBs, and the training requiring a higher RAM memory than the one being provided by colab GPUs, we went with a smaller one.
- Since we needed a small enough dataset for testing purpose. The split of dataset was done in a 5:1 ratio for training data and test data(50000 train data and 10000 test data).

2. Pre-processinga and Feature Engineering

2.1. Normalization

The images of CIFAR10 are represented as 3D arrays representing RGB color pixel value from 0 to 255. The data type of images in CIFAR10 is 'uint8'. This has to be changed to 'float32' due to the fact that the computations involved in neural networks as typically involving floating point numbers such as weights, inputs, outputs etc. and it allows for more precise calculations.

The normalization also involves scaling the 3D array from huge values to a range between 0 and 1. If the conversion to 'float32' is not done then during normalization the division by 255 would lead to integer values which would be incorrect. This step is done in order to better understand the contribution of each pixel value to the pixel.

This step has to be applied for training as well as test data, since we fit the train data to the model, we expect the test data to behave the same way.

2.2. Data Augmentation

Since we are working with image data, the preprocessing we could do is rotating images, shifting image height and width, flipping horizontally and vertically etc. This can be done by using utility class 'DataGenerator' provided by Keras API. We used this library to increase the diversity of the training dataset by applying the above-mentioned transformation to the images in the training set.

When fitting the model with the training data we use the library to generate batches of augmented data and these batches are then used to train the model(SE-ResNet). This enables the model to categorize images based on the diverse set of images it views and learn more robust features thereby generalizing better towards unseen data. This also helps avoid overfitting which we experienced with the CIFAR-10 dataset.

2.3. One-hot encoding

One-hot encoding is a preprocessing technique which transforms categorical variables such as labelled data into binary format ie. 1 or 0. Each category is represented as a binary vector which has only one element corresponding to a specific category is set to 1 and all others to 0.

Here we have a multi-class classification problem as we have images belonging to 10 different classes. The model uses a Softmax activation function at the final layer, which results in a set of probabilities(Probability distribution) of the image belonging to each class. The one-hot encoded representation of the output labels, aligns with the expectations of Softmax where the predicted probability for the correct class is expected to be close to 1, while all other probabilities should be close to 0. This influences the loss function(categorical cross-entropy loss) chosen with the model.

Here we ensure that dummy variable trap problem does not arise as the one hot encoded variables are independent as there is no correlation between the output classes[https://analyticsindiamag.com/when-to-use-one-hot-encoding-in-deep-learning/]. Also the number of categories here are only 10, therefore, we can safely use one-hot encoding on the output labels.

3. Network Structure

3.1. Residual Neural Network 18(ResNet 18)

ResNet is typically used with Convolutional Neural Networks which performs convolution of features over multiple layers. In the earlier CNNs when adding more layers the "Vanishing Gradient" was an issue. The Neural networks that work through backpropogation works with gradient descent algorithms, where the target is to minimize the loss by adjusting the weights. The problem with having multiple layers is that the reduction of gradient would be too pronounced that it disappears and with each added layer and the networks stops learning meaningful patterns from the data. This happens due to gradient of loss function with respect to weights become infinitesimally small during backpropogation. As a result the weights in earlier layers become very small that they hardly update.

Degradation in performance of the network is another problem, where the addition of layers cause the performance to plateau.

One of the solutions to this besides the introduction of various activation functions like variations of ReLU, weight initialization and other regularization techniques as these work for networks with smaller number of convolutional layers, is to change the architecture of the network. This leads to the advent of ResNet introduced by Kaiming He, where we define a residual block which enables the use of "skip connections".

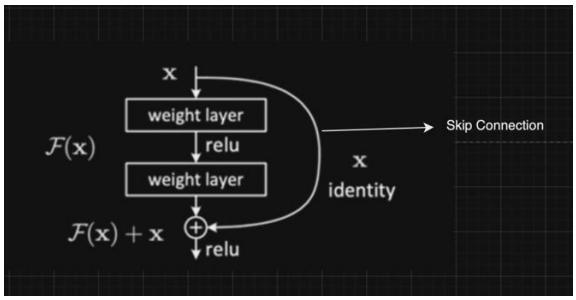

Figure 2. Residual learning block

The concept of propagating input(identity) to the output of a subsequent convolutional blocks is put forth by ResNet architecture. The expression

$$H(x) = F(x) + x \qquad (1.1)$$

in Figure 2. illustrates the key functionality of ResNet, where x is an input to the residual block(identity) and the output from the previous layer and F(x) is the result after convolution over multiple layers(residual mapping learned by the network). The target here is to add the input of the previous layer to the output of the subsequent layers, this is defined as **skip connection**. This allows the gradient to propagate smoothly.

The architecture we formed is ResNet 18, which contains 18 such resnet blocks. The target here was to learn ResNet hence we did not want to over complicate it with 34, 50, or 150 layers.

The output of the residual blocks are passed to a global average pooling that reduces the spatial dimensions. This helps in forming a single value per feature map. In the architecture we have created, the final resnet block has 512 feature maps, on which applying global average pooling leads to 512 values. These values are then passed to a fully connected layer.

3.1.1. ResNet Architecture

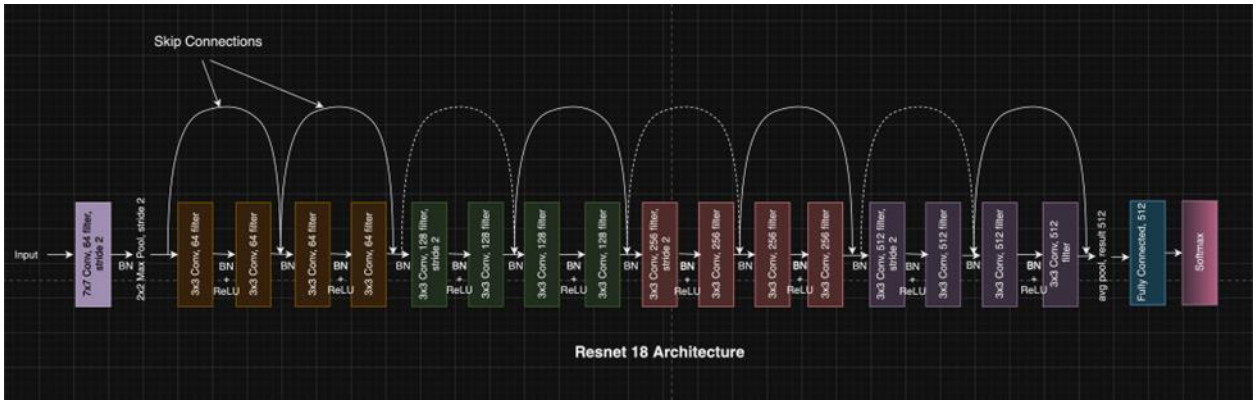

Figure 3. ResNet 18 Architecture

The ResNet 18 architecture illustrated in Figure 3. Here we start with a convolution of the input image with a 7x7 kernel and a stride of 2 and 64 filters or feature maps to reduce input dimensionality by skipping every second pixel in both horizontal and vertical directions of the image, there by achieving a downsampled image for the rest of the network. The 64 filters help capture different patterns in the input image. The output of the convolution is then passed to a Max pooling stage which uses a stride 2, which helps in reducing the dimensions further. This output is passed to the residual blocks.

Here each residual block comprises of 2 convolutional layers. The architecture has four layers of 64 filters, each with a kernel size of 3x3 and 2 skip connections. Then we change the number of filters to 128 with stride 2 in the first layer of the next 4 layers. This is done so that we can reduce the dimensions of the input image, however capturing more details across the filters.

3.1.2. Skip Connections

Here the dotted lines for skip connections denote the increase in features captured since we increase the number of filters. For instance, the first dotted line denotes, the output of the 64 filters when passed to the output of subsequent layer, it has to match the dimension before they are combined as mentioned in (1.1). This dimension matching is done by the concept of "same" padding where we pad the input with 2 rows of zeros for increasing the size of the input in order to match the size of the output from the 128 filter. When doing this downsampling of the input image(reducing the width and length of the image) we use a different sized kernel ie. 1x1 kernel with a stride of 2. The stride helps in reducing the dimension of the input image. The 1x1 kernel is used here to reduce the computational cost and memory requirements effectively reducing the dimensionality without introducing additional parameters. This behaviour of reducing the dimension is maintained throughout the network whenever we increase the filter size.

3.1.3. Softmax

- Once the Residual blocks have been traversed the feature map is passed through average pooling layer which calculates the average output of each feature map(filter) across all spatial locations in the previous layer which produces one output per feature map ie. 512 outputs(which is alternative to flattening the feature map).
- Flattening is not preferred here due to the fact that we would lose important features. These outputs are then passed through a fully connected layer containing 512 neurons with Softmax activation.
- The Softmax helps transform these output into vector of probabilities and produce a probability distribution over the input classes.
-

3.1.4 Batch Normailization

- As per Ioffe and Szegedy(2015), batch normalization can be used to normalize inputs to non-linear activation functions. We use ReLU in between the each of the two residual blocks with the same channels(ie. 64, 64, 128, 256, 512).
- Here each of the call to 64 channels is composed of 2 layers in the defined network, and between each of these we apply Batch Normalization and pass it to a ReLU activation to introduce non-linearity to capture non-linear details from the image.

*"Batch Normalization seeks a stable distribution of activation values throughout training, and normalizes the inputs of a nonlinearity since that is where matching the moments is more likely to stabilize the distribution"*Ioffe and Szegedy(2015)

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, None, None, 64)	9408
batch_normalization (BatchNormalization)	(None, None, None, 64)	256
max_pooling2d (MaxPooling2D)	(None, None, None, 64)	0
resnet_block (ResnetBlock)	(None, None, None, 64)	74564
resnet_block_1 (ResnetBlock)	(None, None, None, 64)	74564
resnet_block_2 (ResnetBlock)	(None, None, None, 128)	232584
resnet_block_3 (ResnetBlock)	(None, None, None, 128)	297608
resnet_block_4 (ResnetBlock)	(None, None, None, 256)	928016
resnet_block_5 (ResnetBlock)	(None, None, None, 256)	1189136
resnet_block_6 (ResnetBlock)	(None, None, None, 512)	3707424
resnet_block_7 (ResnetBlock)	(None, None, None, 512)	4753952
global_average_pooling2d_8 (GlobalAveragePooling2D)	(None, 512)	0
dense_16 (Dense)	(None, 10)	5130

Figure 4. Summary of the Network

The layers and the number of parameters are shown as the summary in Figure 4.

3.2. SE Block

We used SE module because the SE module seamlessly integrates into existing network architectures.

- SE blocks are better than other methods for fixing features because they work well, are easy to understand, and can fit into Resnet model without much changing. SE blocks are like smart tools used by computers to understand images better. They work by focusing on important parts of the picture while ignoring less important ones. Imagine looking at a photo and automatically zooming in on the most relevant details.
- These blocks are popular because they're not too complicated to use and fit neatly into existing image analysis systems without causing major changes. So, they're like handy add-ons that make computer vision systems smarter and more accurate at recognizing objects in pictures. They make models better at understanding images by focusing on important parts and making fewer mistakes.

- In our model implementation, the SE block is integrated within each residual block of Resnet 18 and thus ensures that the benefits of feature recalibration are consistently applied across the entire network, maximizing its effectiveness in learning complex patterns and enhancing classification accuracy. Mainly focusing on channel correlations, the SE module makes only a slight increase in computational overhead while delivering improved outcomes.
- The entire running time was almost same but the accuracy got increased by 3%(from 79% to 82%).

3.2.1 SENet Architecture

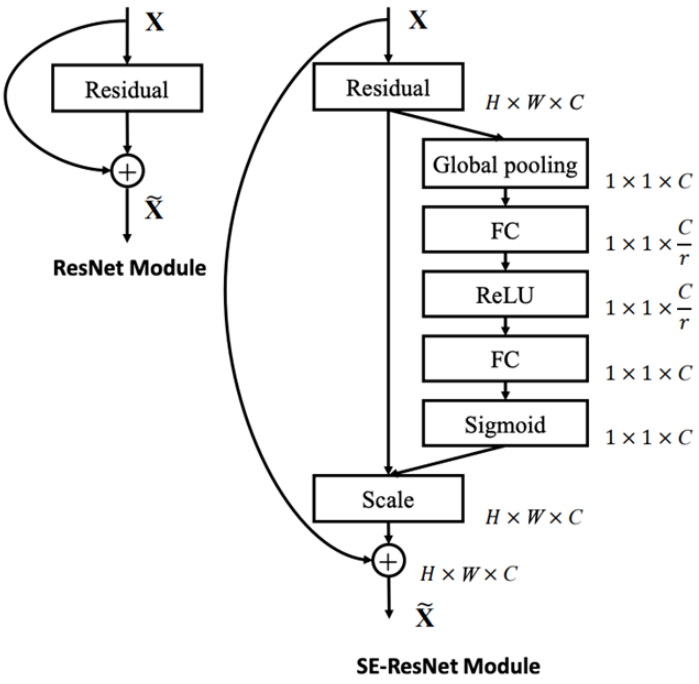

Figure.5 Architecture of SE Block

Figure.5 describes the architecture of SE Block. The SE block consists of two main stages: the "squeeze" stage and the "excitation" stage.

3.2.2. Squeeze Stage:

- The squeeze operation makes the input feature maps smaller and combines information from different channels. It does this by using global average pooling, which creates a tiny map for each channel. This is achieved by using global average pooling, which produces a 1x1 feature map for every channel.

3.2.3. Excitation Stage:

- In the excitation stage, the channel-wise summary from the squeeze stage passes through fully connected layers to learn channel-wise importance scores. ReLU

activations add non-linearity to capture complex relationships between channels. Subsequently, a sigmoid activation function T ensures that the computed scaling factors are normalized between 0 and 1, facilitating meaningful modulation of feature responses. These scores modulate the importance of each channel by element-wise multiplication with the original feature maps

In our architecture, the SE block plays a crucial role in refining the representation of features within the model. It achieves this by first compressing the spatial dimensions of the feature maps to a single value for each channel, effectively condensing the information. This compression results in a 1x1 feature map for each channel, where the number of channels remains unchanged.

- By introducing the SE block after this normalization step, we maximize its effectiveness by working with standardized feature maps.
- Overall, the integration of the SE block at this stage serves to enhance the model's representational power.
- By recalibrating the feature responses on a channel-wise basis, the SE block enables the model to focus more effectively on relevant features during training. This targeted recalibration contributes to improved model performance and better generalization across diverse datasets and tasks.

4. Hyperparameters

4.1. Learning Rate Scheduler

- The Learning Rate Scheduler is another functionality provided by Keras.callbacks, which helps us tune the learning rate of the model after each epoch. Here we have used Default learning rate(LR) as LR = $1e^{-3}$ and then we use scheduler with respect to epochs like

  LR = $1e^{-1}$ when the epoch is less than or equal to 15

  LR = 1 e^{-2} when epoch is greater than 15 and less than 20

4.2. Reduce Learning Rate on Plateau

- Another callback functionality provided by Keras.callbacks is ReduceLROnPlateau, where we specify the factor by which the learning rate has to be adjusted, number of epochs after which it has to change the learning rate provided no improvement is made, a minimum learning rate below which it should not go, and number of epochs to wait before resuming normal operation after learning rate has been reduced.

4.3. Batch Size :

- We used a batch size of 100 in our training process. We also tried with 32 and then 64.
- Larger batch sizes enable the model to simultaneously process more examples, thereby potentially accelerating training times on contemporary hardware. Moreover, larger batches tend to produce more stable gradients, fostering smoother convergence during the training phase.

4.4 Number of Epochs :

The choice of epochs in neural network training defines the frequency of learning iterations from the dataset, ensuring a balance between avoiding overfitting and underfitting.

- Too few epochs may cause underfitting, where the model overlooks important patterns in the data. Conversely, too many epochs can lead to overfitting, where the model memorizes the training data but struggles to generalize to new examples
- To optimize computational efficiency and availability of GPU, we have selected 25 epochs for each of the 5 folds in cross-validation, aiming to reduce training time while enhancing model performance.

5. Cost / Loss / Error / Objective Function

In machine learning, the concept of "loss" is important and it is the deviation between predicted and actual values. This discrepancy is quantified through a "loss function," a crucial metric during the training phase. Often used interchangeably, "cost function" and "loss function" refer respectively to the error for a single training example and the average of loss functions across an entire training dataset.

- In the context of a multi-class, single-label problem, there are two primary methods for calculation: Categorical Cross-Entropy and Sparse Categorical Cross-Entropy.
- Categorical cross-entropy loss, also referred to as softmax loss, involves obtaining predicted probabilities by applying a softmax function to the model's outputs. This function transforms the raw outputs into a probability distribution across the target classes.
- To compute categorical cross-entropy loss, the negative logarithm of the predicted probability (as shown below in (1.2))assigned to the correct class label is taken. This loss is then summed across all training examples and averaged to derive a scalar value representing the overall model performance.

$$L = -(1/m) \sum^{m}_{i=1} y_i * \log(y\hat{}_i)$$

(1.2)

- Unlike categorical cross-entropy loss, which requires true labels to be represented as one-hot encoded vectors, sparse categorical cross-entropy loss operates with target labels directly as integers indicating class indices. Internally, this loss function converts the true labels into one-hot encoded vectors and then proceeds with the regular categorical cross-entropy loss calculation.
- Given that our data is already in a one-hot encoded format, we have opted for categorical cross-entropy as the loss function in this scenario, as it aligns better with the representation of our target labels and facilitates the training process effectively.

6. **Optimizer**

Optimizers are like special tools in a math toolbox for adjusting how a neural network learns. They use gradients and other info to tweak the network's weights. It actually act as a guide that help the network move in the right direction

- In this setup, we experimented with Adam and Nadam Optimizer
- Adam optimizer is the combination of momentum and RMSProp optimizers.
- Nadam, which builds upon Adam by adding Nesterov momentum, Nadam's advantage lies in its ability to handle rapid convergence while maintaining stability across different learning rates. This feature helps in reducing oscillations and overshooting, potentially leading to faster convergence compared to Adam alone.
- Therefore, for our ResNet architecture on the CIFAR-10 dataset, Nadam emerged as a reliable and efficient choice.

7. **Cross Fold Validation**

- In our architecture, we used K-fold cross-validation
- In K-fold cross-validation, we specify a value for K, dividing the dataset into K partitions of equal size. K - 1 partitions are used for training, while one partition is reserved for testing. This process is repeated K times, with a different partition used for testing each time.
- For our experiments, we chose K = 5, which corresponds to an 80/20 split repeated 5 times. In each iteration, the same model is trained and evaluated on the testing partition. The performance metrics are then averaged across all folds to provide a more reliable estimate.
- However, employing K-fold Cross-Validation also increases the computational cost since the model needs to be trained K times. In the context of the CIFAR-10 dataset and ResNet models, K-Fold Cross-Validation is often preferred over other methods due to its effectiveness in evaluating model performance while mitigating certain limitations.
- Here for CIFAR-10, which consists of 60,000 32x32 color images across 10 classes, K-Fold Cross-Validation helps in utilizing the available data more effectively by

ensuring that each sample is used for both training and validation exactly once. This is crucial for a dataset like CIFAR-10, where having a diverse and representative training set is essential for building a robust model. K-Fold Cross-Validation allows for better estimation of model performance by reducing the variance associated with a single train-test split.

8. **Results**

Accuracy Precision and Recall

Given below in Figure.6 and Figure. 7 shows the precision and accuracy and recall and f1 score of each classes.

	precision	recall	f1-score	support
0	0.81	0.85	0.83	1000
1	0.87	0.90	0.88	1000
2	0.82	0.69	0.75	1000
3	0.71	0.55	0.62	1000
4	0.78	0.79	0.79	1000
5	0.75	0.69	0.72	1000
6	0.77	0.92	0.84	1000
7	0.82	0.87	0.85	1000
8	0.91	0.89	0.90	1000
9	0.79	0.89	0.84	1000
accuracy			0.80	10000
macro avg	0.80	0.80	0.80	10000
weighted avg	0.80	0.80	0.80	10000

Figure 6. Classification report

Class	Precision	Recall	Accuracy
airplane	0.808389	0.848	0.848
automobile	0.868142	0.902	0.902
bird	0.815077	0.692	0.692
cat	0.705882	0.552	0.552
deer	0.7833	0.788	0.788
dog	0.747573	0.693	0.693
frog	0.771429	0.918	0.918
horse	0.821866	0.872	0.872
ship	0.909651	0.886	0.886
truck	0.789849	0.887	0.887

Figure 7. Precision Recall and accuracy

Confusion matrix is illustrated in Figure 8.

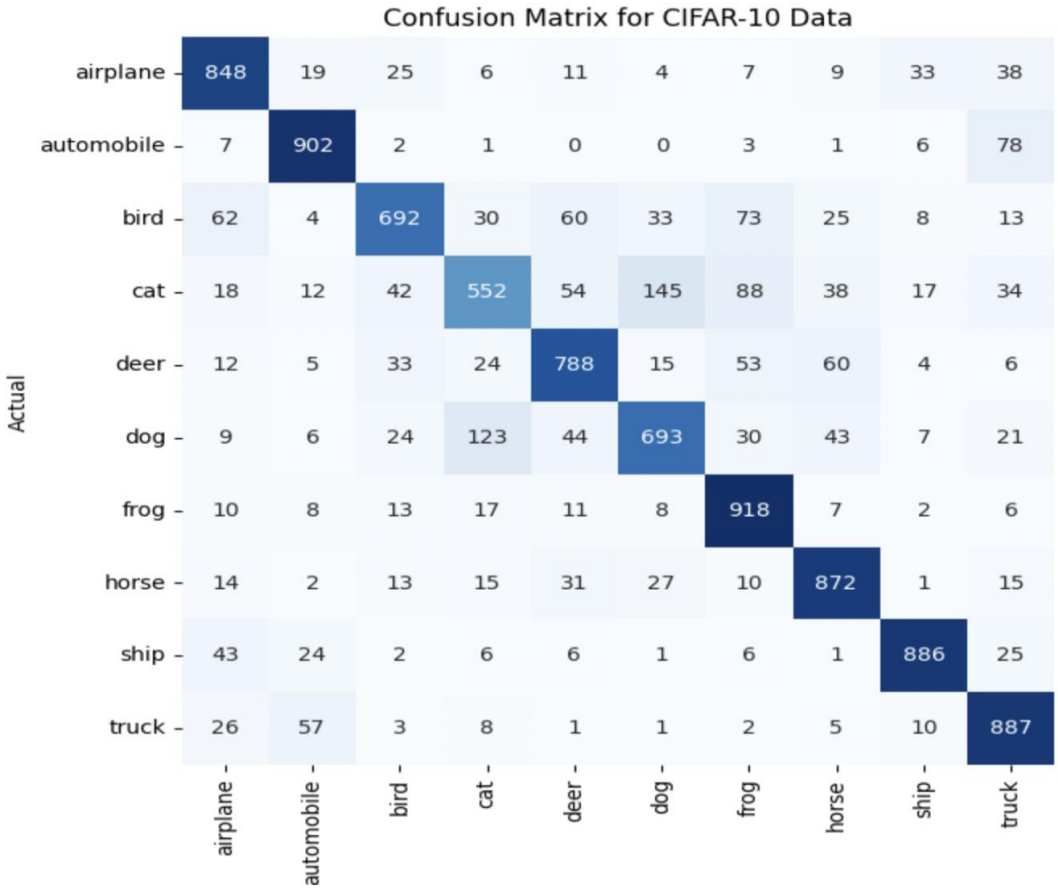

Figure 8. Confusion Matrix

Figure 9 shows the Loss and accuracy of validation data in K(5)-Fold Cross validation and in Figure 10 and Figure 11 illustrates the Accuracy vs epoch for each fold and Loss vs epoch for each fold respectively.

```
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 0.5512568354606628 - Accuracy: 81.27999901771545%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.5562928915023804 - Accuracy: 81.48000240325928%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.5466974377632141 - Accuracy: 81.36000037193298%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.5625331401824951 - Accuracy: 80.94000220298767%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.54581618309021 - Accuracy: 81.11000061035156%
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 81.23400092124939 (+- 0.18990510211388273)
> Loss: 0.5525192975997925
------------------------------------------------------------------------
```

Figure 9. Loss and Accuracy in each Fold

Figure 9 shows the loss and accuracy in each fold and their average accuracy and loss for all folds

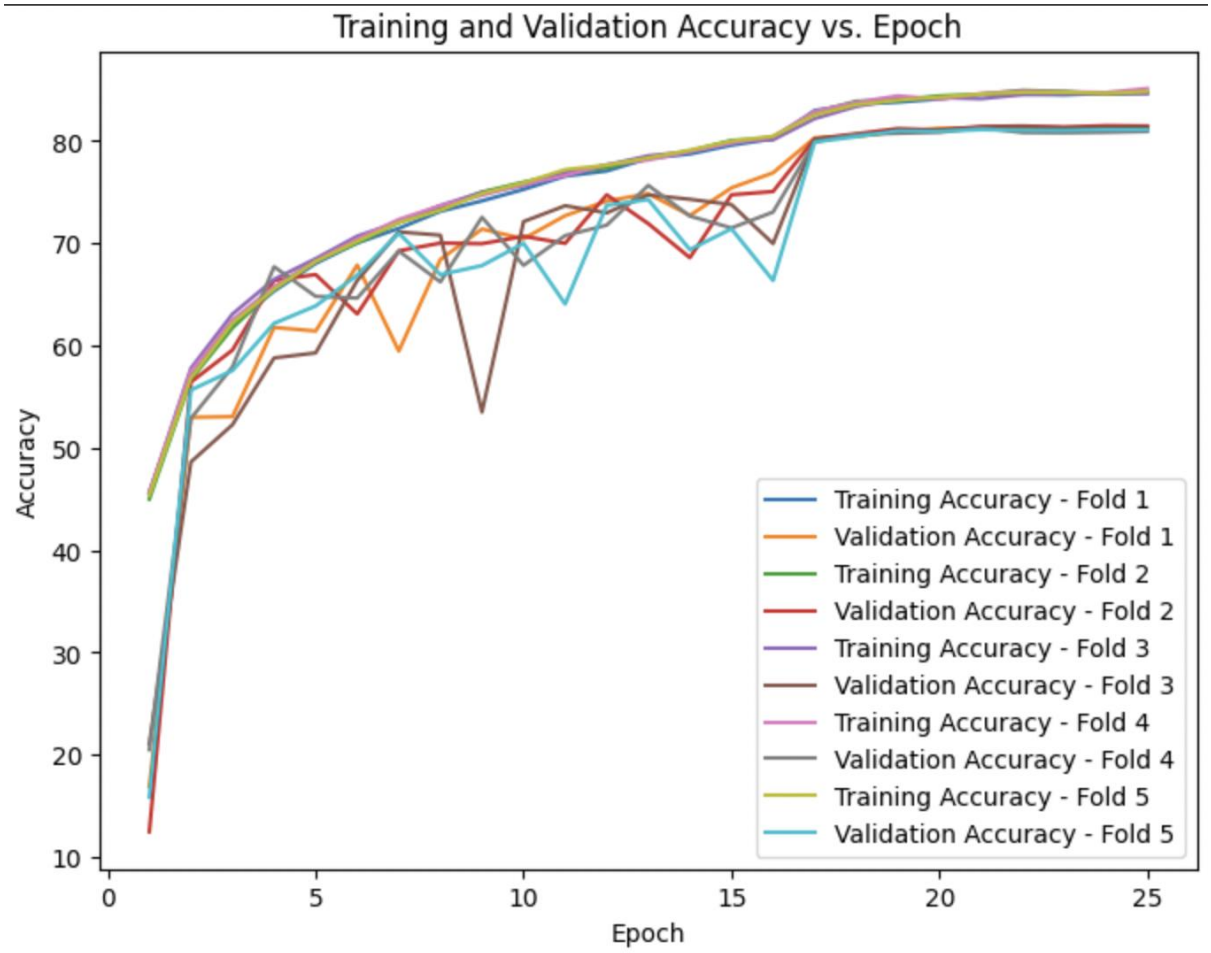

Figure 10. Accuracy vs epoch

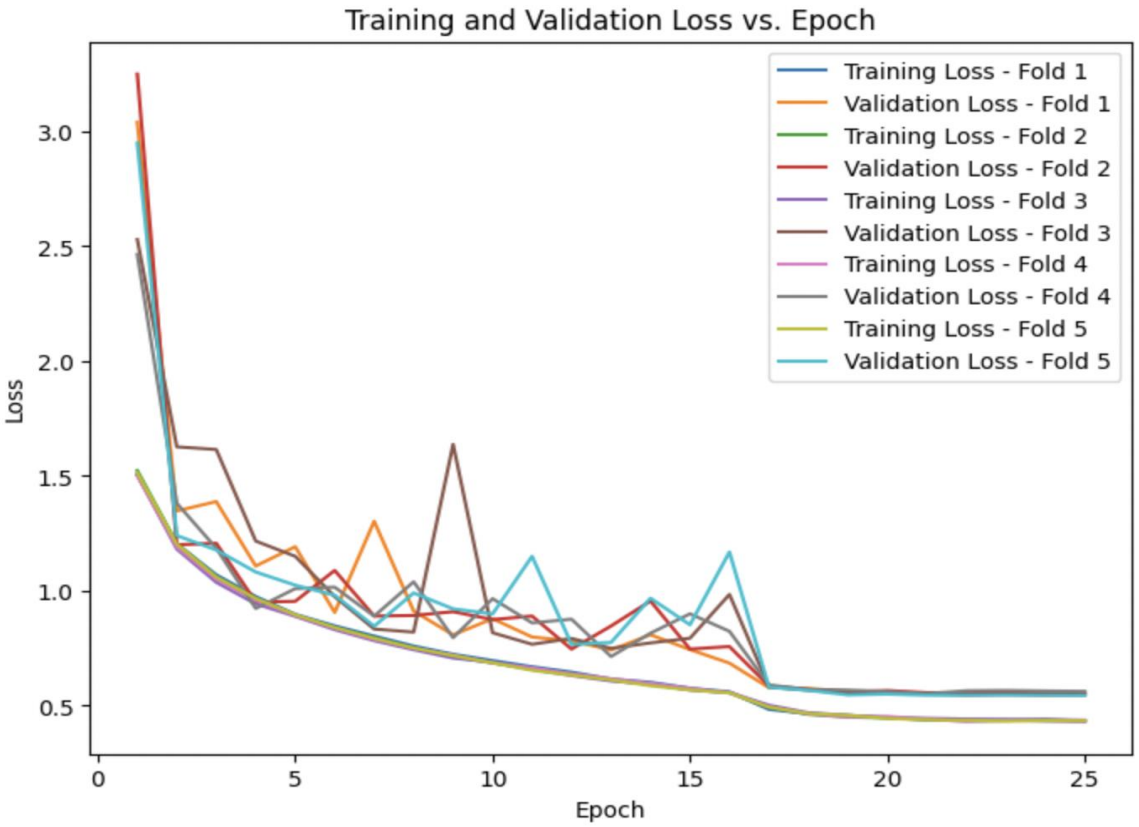

Figure 11. Loss vs epoch

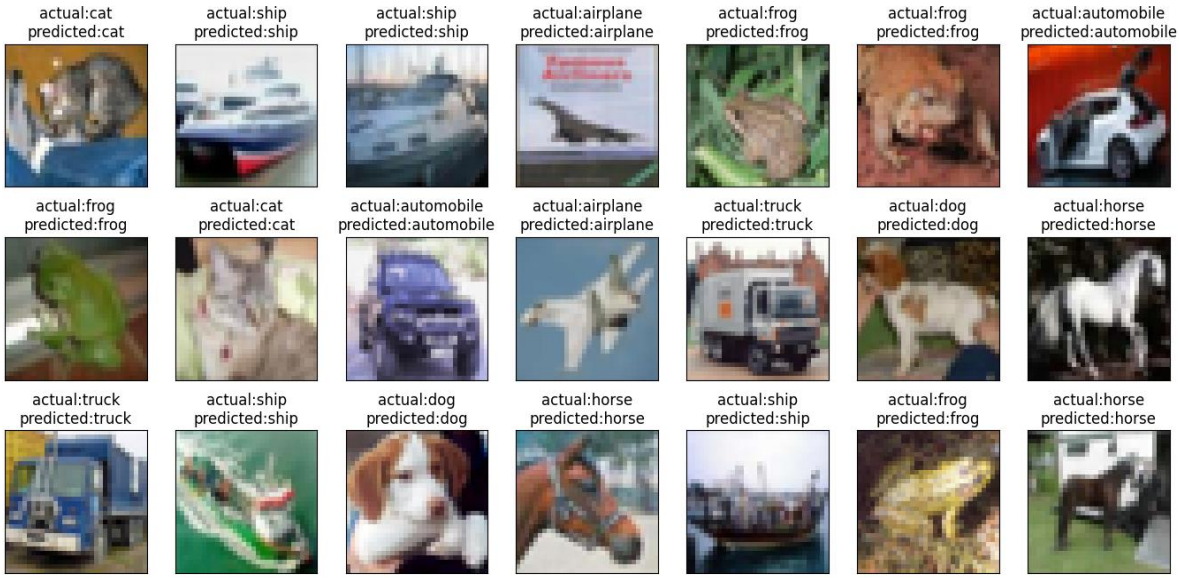

Figure 12. Visualization of result of some samples from dataset

Figure 12. shows the result of some of the predictions and some of the actual results. Here we see most of the images are classified correctly, which illustrates the accuracy of 81%.

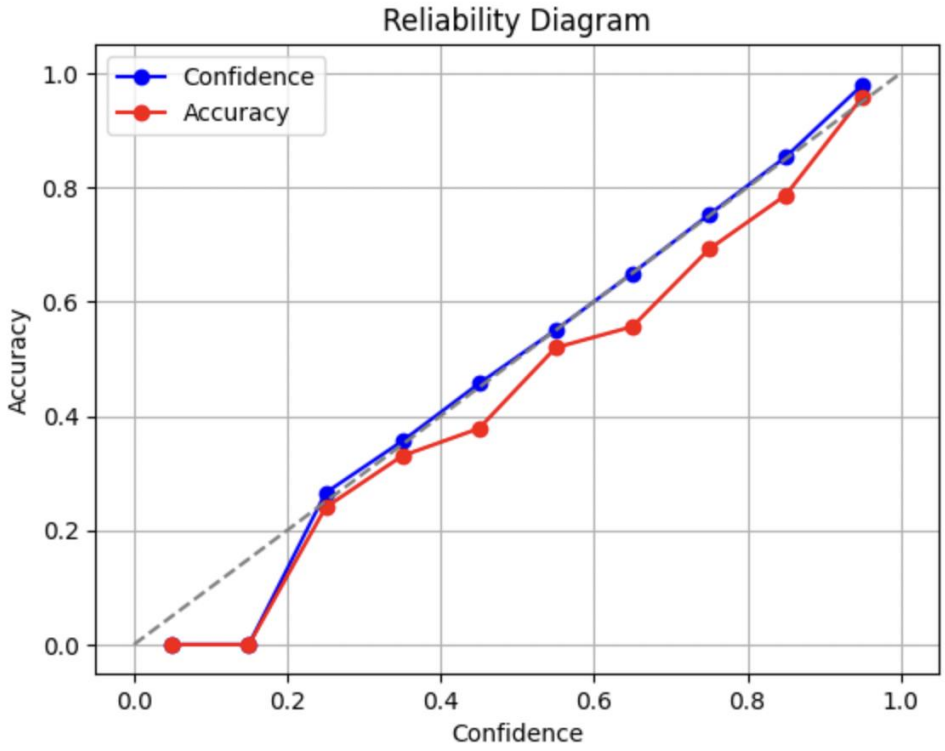

Figure 13. Reliability Diagram

9. **Evaluation of the Results**

The objective here was to create a Convolutional Neural Network(CNN) architecture to classify CIFAR10 dataset. We created a SE-ResNet with 18 blocks which provides a reasonable accuracy of 81%.

- Here we had taken ResNet18 to streamline computational efficiency thereby reducing the execution time on the GPU provided by Google Colab.
- We have a SE Net block within each residual block which increases accuracy of the neural network about 3% from our observations.
- The Loss vs. Epoch and Accuracy vs. Epoch plots in Figure 10 and Figure 11 shows that the model generalizes well to test data for the SEResNet18 network created as the difference between the train and test data is close to convergence.
- This also shows us that the model does not overfit to the training data by memorizing the results
- In the Figure 6, we note that the accuracy of 81% for the test data with precision and recall. Precision indicates the accuracy of positive predictions made by the model, and the recall indicates how many of the predicted positives are actually positive. The value of recall obtained is 81. High recall indicates low false negative rate. ie. The predicted class would not have many false negative errors.
- Confusion matrix shown in Figure 8 highlights the misclassifications between the various labels and inputs such as 57 instances of truck being incorrectly labeled as

automobile. This tells us that the dataset has images that can be misclassified. The image class that is mostly misclassified are Cats and Dogs, 145 instances of Cats are misclassified as Dogs, and the least misclassified classes are Frog.

- The Reliability diagram shown in Figure 13 represents a scatter plot of confidence on X-axis and accuracy on Y-axis, which tells us how well the predicted probability of the model aligns with the actual probability of the predicted class. Ideally, the curve should follow the dashed grey line(diagonal from bottom-left to top-right). This figure provides insight into the reliability of the model. Deviation from the grey line indicates where the model becomes too confident or where its less confident with the predictions.
- The Reliability diagram is formed by :
    - Creating 10 different bins based on the predicted probabilities of each sample image. Each bin represents a range of predicted probabilities, where we define a range from 0 to 1 separated by equal intervals.
    - Then we iterate through each bin and check if the maximum predicted probability belongs to the current bin. ie. If the image belongs to the current bin.
    - Number of samples in each bin is recorded and we calculate the accuracy within the bin by comparing the ground truth labels with the predicted labels, and compute the confidence within the bin by averaging the maximum predicted probabilities for each sample

10. Impact of Varying Parameters/Hyperparameters

10.1. Overfitting

- Overfitting occurs when the model memorizes the data and predicts the result with the memory of the trained data rather than learning underlying patterns from the features. As a result, the model gives good results for the training data, however, fails to generalize well for the test data. This can be observed from the accuracy vs epoch graph as illustrated in Figure 14.
- The higher training accuracy but the low test accuracy indicates that the generalization to new input fails to classify the results accurately.
- Another possibility is that the model becoming too complex for the dataset, which is not the case here as we have a ResNet18.
- This behaviour can also be exhibited if the model has a high variation in the number of outliers.
- In the Loss vs. Epoch graph 14, we observe that both the training and testing loss decrease where the model learns patterns in the data. However, later in the epochs the divergence in the plot between training and testing loss is large, whereby the training loss continues to decrease but the test loss plateaus.
- This is overcome by adding the regularization by using ImageDataGenerator which in addition to augmenting the training data such as rotating, shifting, etc. the image,

whereby it introduces variability, it acts as an indirect regularizer by introducing noise and variation to the training data

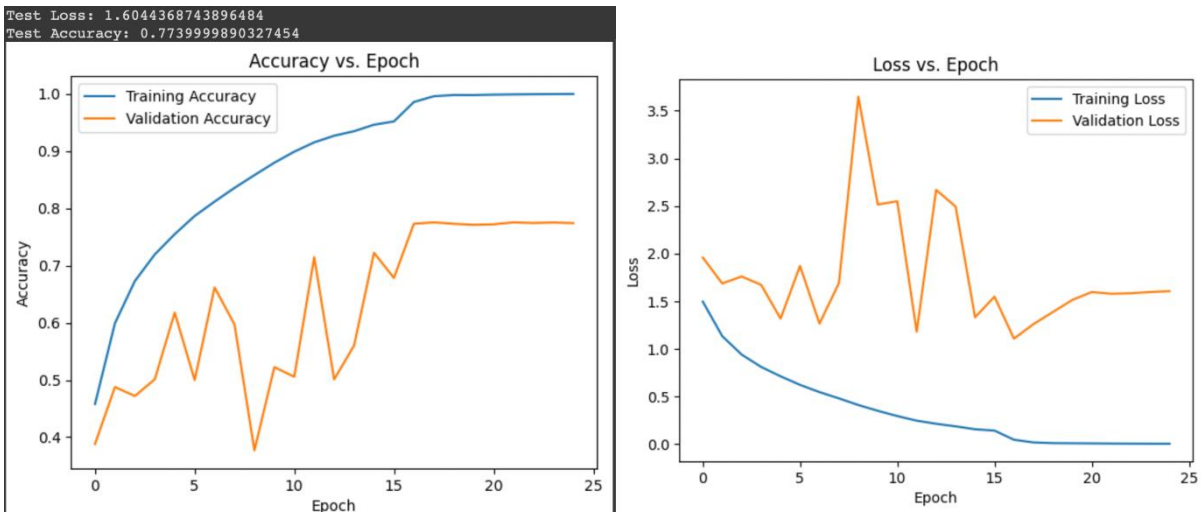

Figure 14. Accuracy vs. Epoch and Loss vs. Epoch for Overfitting

The plot shown in Figure 14. Illustrates the clear demonstration of Overfitting where the Training accuracy keeps on decreasing however the Validation accuracy plateaus at 77.39 % and the train accuracy goes to a 100. This clearly states that the model is memorizing the training data and is classifying well with respect to the training data, however, fails to classify the test data will the same precision, which is a clear characteristic of Overfitting.

	precision	recall	f1-score	support
0	0.76	0.78	0.77	1000
1	0.81	0.82	0.82	1000
2	0.66	0.61	0.63	1000
3	0.51	0.53	0.52	1000
4	0.67	0.67	0.67	1000
5	0.60	0.60	0.60	1000
6	0.76	0.80	0.78	1000
7	0.77	0.75	0.76	1000
8	0.82	0.83	0.83	1000
9	0.79	0.77	0.78	1000
accuracy			0.71	10000
macro avg	0.71	0.71	0.71	10000
weighted avg	0.71	0.71	0.71	10000

Figure 15. Classification Report for Overfitting

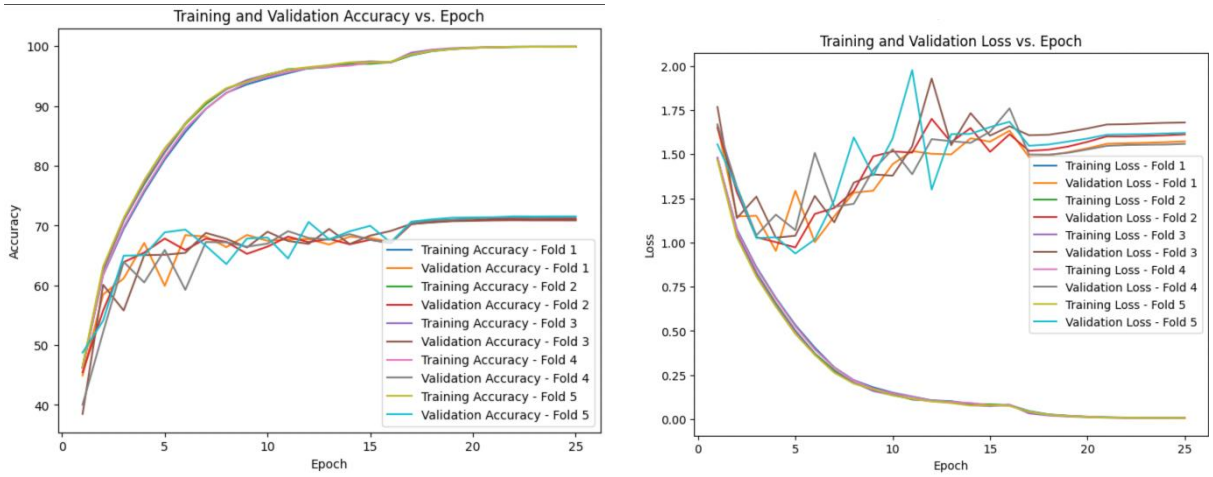

Figure 16. Accuracy vs. Epoch and Loss vs. Epoch for Overfitting with Cross Validation for Overfitting without Batch Normalization

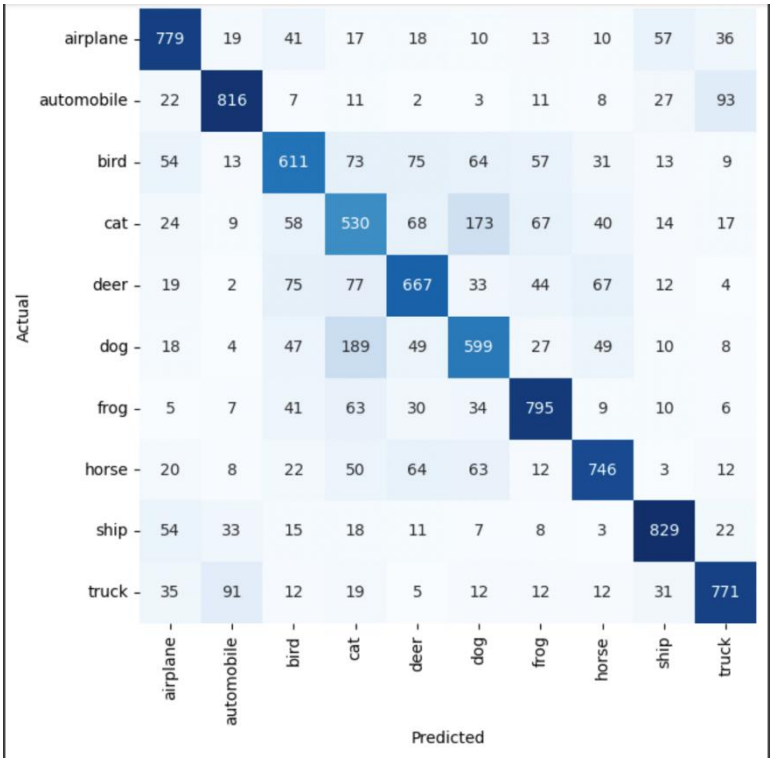

Figure 17. Confusion Matrix for Overfitting

The figure 15. Shows the classification report for the overfitted model with the removal of batch normalization. This illustrates the importance of batch normalization in the network. The accuracy decreased to 71% which can be read from the confusion matrix where the dog class is often misclassified as cat class over 189 instances. There are a lot of other misclassifications as well highlighted in the confusion matrix.

10.2 Learning rate

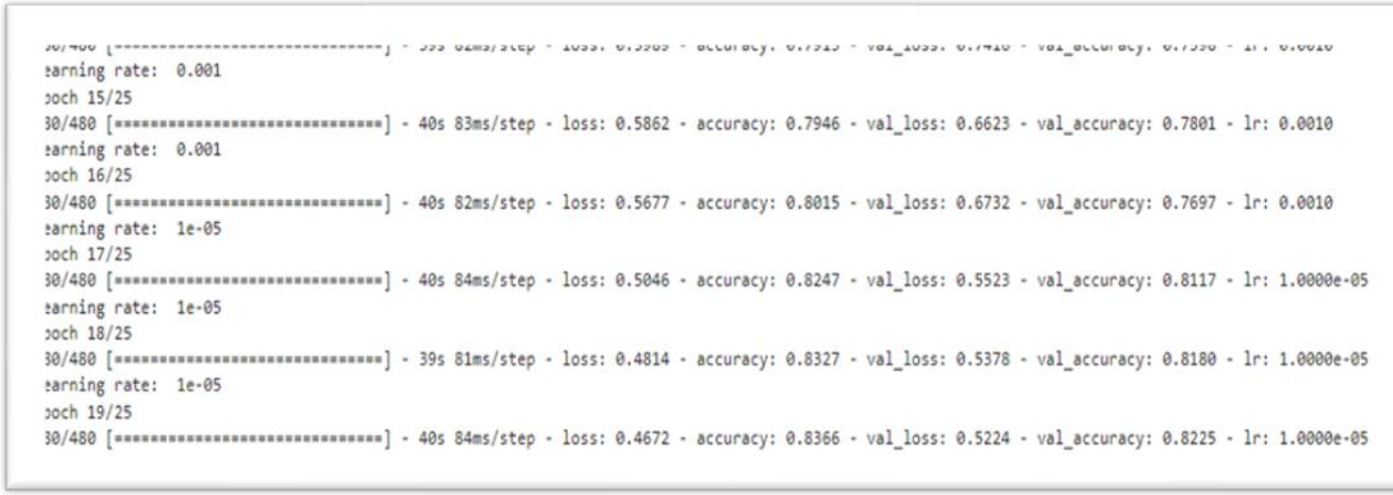

Figure 18. Accuracy changing when Learning Rate changes

As the learning rate rises, reaching LR = 1e-1 when the epoch is 15 or more, we observe a notable improvement in accuracy. In Figure 18 presented above, the accuracy climbs from 76% to 81% as the learning rate increases. This underscores the significant impact of learning rate adjustments on model performance.

10.3. Optimizer

Initially, we experimented with the Adam optimizer, and subsequently, we tried Nadam. The transition to Nadam yielded a modest but noteworthy increase of 1% in accuracy. This improvement may be attributed to the enhanced convergence properties and momentum adjustment provided by Nadam compared to Adam.

10.4. Batch Size

Initially, we experimented with a batch size of 32, and later we tried 64. We noticed that larger batch sizes speed up training but also increase computational demands. So, we settled on a batch size of 100 for a balance between efficiency and training time.

10.5. Number of epoch

As the number of epochs increases, there is a corresponding rise in accuracy. However, due to limited GPU availability, we had to restrict the epochs to 25 within a 5-fold cross-validation setup. Initially, we conducted experiments without fold to assess each parameter's performance more comprehensively.

11. **Reference**

1. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). DOI: 10.1109/CVPR.2016.90
2. Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning (Vol. 37, pp. 448-456).
3. kaggle.com. (n.d.). Image classification of CIFAR-10 using TensorFlow. [online] Available at: https://www.kaggle.com/code/viratkothari/image-classification-of-cifar-10-using-tensorflow [Accessed 10 Mar. 2024].
4. kaggle.com. (n.d.). Image Classification CNN With resNet. [online] Available at: https://www.kaggle.com/code/rizkiprof/image-classification-cnn-with-resnet [Accessed 10 Mar. 2024].
5. Modi, R. (2022). ResNet — Understand and Implement from scratch. [online] Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/resnet-understand-and-implement-from-scratch-d0eb9725e0db.
6. Bhandari , A. (2020). Keras ImageDataGenerator for Image Augmentation | Python Use Case. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/.
7. Moreno, F.A. (2021). Sparse Categorical Cross-Entropy vs Categorical Cross-Entropy. [online] Medium. Available at: https://fmorenovr.medium.com/sparse-categorical-cross-entropy-vs-categorical-cross-entropy-ea01d0392d28.
8. ht Chand, S. (2023). Choosing between Cross Entropy and Sparse Cross Entropy — The Only Guide you Need! [online] Medium. Available at: https://medium.com/@shireenchand/choosing-between-cross-entropy-and-sparse-cross-entropy-the-only-guide-you-need-abea92c84662 [Accessed 13 Mar. 2024].
9. paperswithcode.com. (n.d.). Papers with Code - SENet Explained. [online] Available at: https://paperswithcode.com/method/senet [Accessed 13 Mar. 2024].
10. GitHub. (n.d.). machine-learning-articles/how-to-use-k-fold-cross-validation-with-keras.md at main · christianversloot/machine-learning-articles. [online] Available at: https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-validation-with-keras.md [Accessed 13 Dec. 2023]
11. kaggle.com. (n.d.). Simple Keras Model with k-fold cross validation. [online] Available at: https://www.kaggle.com/code/stefanie04736/simple-keras-model-with-k-fold-cross-validation [Accessed 19 Mar. 2024].
12. Stack Overflow. (n.d.). Sklearn StratifiedKFold: ValueError: Supported target types are: (‘binary’, ‘multiclass’). Got ‘multilabel-indicator’ instead. [online] Available at: https://stackoverflow.com/questions/48508036/sklearn-stratifiedkfold-valueerror-supported-target-types-are-binary-mul [Accessed 19 Mar. 2024].