## 1. Introduction

This notebook presents the implementation deep learning model (Convolutional neural network) for "CIFAR10" image classification. The aim is to get the highest test accuracy using the specific given model. The problem of this image classification model is to correctly classify each image into one of the ten classes, which include trucks, frogs, horses, deer, dogs, cats, birds, ships, and cars.

## 2. Read dataset and Create Data loaders

The CIFAR-10 dataset consists of 60000 32x32 color images divided into 10 classes. PyTorch's torchvision.datasets module can be used to download the dataset. To prepare the dataset for training a CNN model, data transformation and data loaders are created. To increase the training set and reduce overfitting, the transformation pipeline for the training dataset includes random cropping size of 32 with padding of size 4 on each side, random horizontal flipping, and normalisation. For the test dataset, only normalization is performed to maintain consistency across both datasets. Data loaders are created for training and test datasets with a batch size of 128 to ensure faster training since larger batch size can lead to faster training [1].

## 3. Implementing Model

The model consists of backbone and classifier and is implemented according to following diagram Fig 1.
The Backbone model has 6 blocks, each with 3 convolutional layers, a fully connected layer, batch normalization, and ReLU activation. The 3 convolutional layers in each block have different kernel sizes and their output is multiplied by a vector generated by an MLP before being summed and passed through a ReLU activation. Spatial average pooling layer is used to reduce the spatial dimensions of the feature maps and prevent overfitting. MaxPooling layer with kernel size of 2x2 and stride 2 is applied after 2nd, 4th, and 5th block.
For the CIFAR10 dataset, a combination of different kernel sizes is used to capture channel-wise and spatial information while remaining computationally efficient [2]. The softmax classifier consists of Flattening, Linear, ReLU activation, Batch normalization, Dropout, Linear, and Softmax layer. The Softmax layer produces a probability distribution over 10 classes, and the class with the highest probability is taken as the predicted class label.

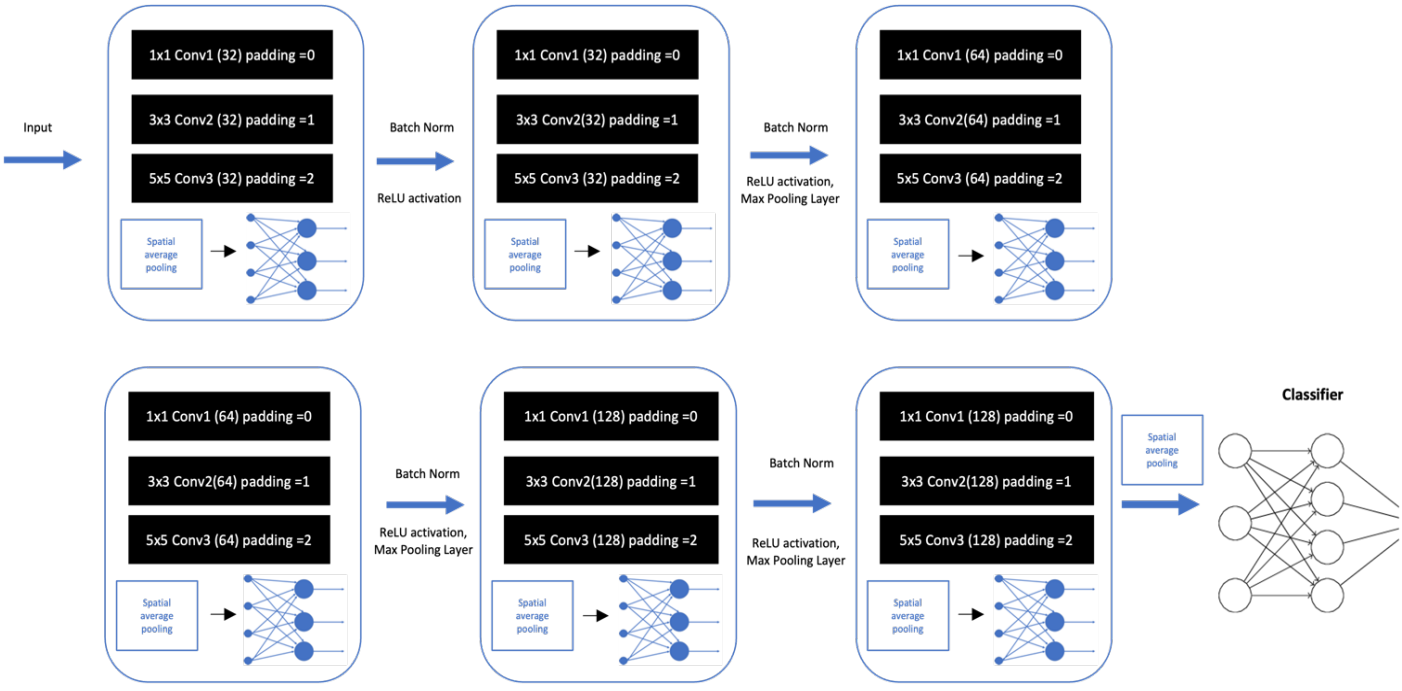


**Fig 1: Model architecture**

## 4. Training, Loss function, Initialization method, Optimizer and Hyperparameter Tuning

The cross-entropy loss function is used to measure the difference between the predicted probabilities and the true labels. The model is initialized using default initialization which is Kaiming, which works well with activation functions like ReLU. The optimizer used is stochastic gradient descent (SGD) with different learning rates of 0.01, 0.001 and 0.001, and a weight decay value of 5e-4 to prevent overfitting [3]. The momentum parameter of 0.9 is used

to speed up the convergence of the optimization algorithm. The training and test accuracy/loss are stored in empty lists for each epoch and all steps. The training script loops through each batch in the data loader, performs forward pass and backpropagation to update the model's parameters based on gradients using optimizer, and records the training accuracy and loss for each epoch and step. The testing script evaluates the model's performance on the test dataset without updating the parameters. It loops through each batch, computes loss, and determines the predicted class for each image.

As mentioned, 3 different learning rate is compared. It seems that a learning rate of 0.01 (model 1) was found to perform better than 0.001 and 0.005 after experimenting with different values. A lower learning rate such as 0.001 or 0.005 may require more iterations to converge. For 150 epochs, test accuracy for each model is shown in Table 4. Thus, the model with learning rate of 0.01 is selected and trained. In the following, Curves for evolution of loss and accuracy are shown as well as test performance using a heatmap.

**Table 1. Model 1**

| Hyperparameters | Values |
|---|---|
| Batch size - Training set | 128 |
| Batch size - Test set | 128 |
| Learning rate | 0.01 |
| Weight decay | 5e-4 |
| Momentum | 0.9 |
| Number of epochs | 150 |
| Training steps per epoch | 391 |
| Test steps per epoch | 78 |

**Table 2. Model 2**

| Hyperparameters | Values |
|---|---|
| Batch size - Training set | 128 |
| Batch size - Test set | 128 |
| Learning rate | 0.005 |
| Weight decay | 5e-4 |
| Momentum | 0.9 |
| Number of epochs | 150 |
| Training steps per epoch | 391 |
| Test steps per epoch | 78 |

**Table 3. Model 3**

| Hyperparameters | Values |
|---|---|
| Batch size - Training set | 128 |
| Batch size - Test set | 128 |
| Learning rate | 0.001 |
| Weight decay | 5e-4 |
| Momentum | 0.9 |
| Number of epochs | 150 |
| Training steps per epoch | 391 |
| Test steps per epoch | 78 |

**Table 4. Test accuracy**

| | |
|---|---|
| Model 1 | 89.55 % |
| Model 2 | 87.56 % |
| Model 3 | 87.44 % |

### 4.1. Curves for evolution of loss and accuracy:

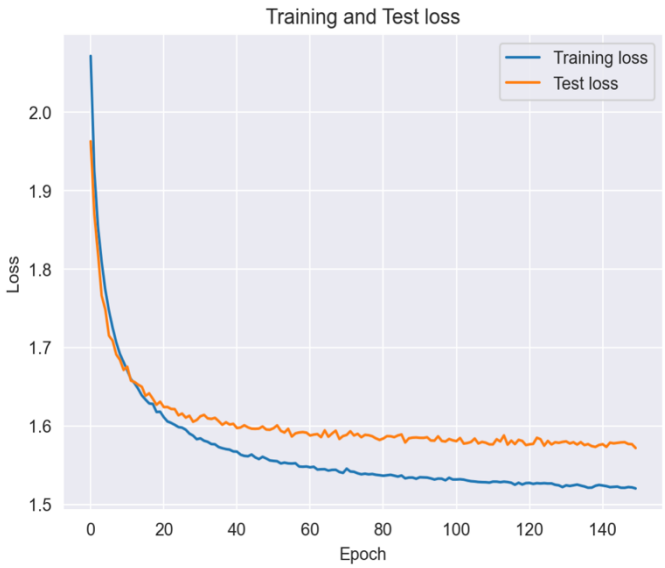


**Fig 2: Training and Test Loss**


**Fig 3: Training and Test Accuracy**

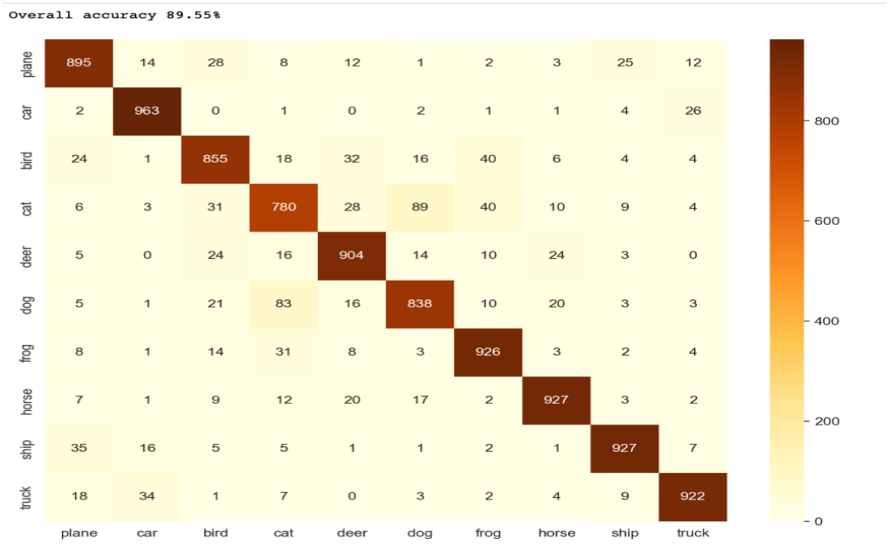


**Fig 4: Test performance**

## 5. Main findings and Conclusion:

**(1)** Comparison of 5 epoch training time for a neural network: Google Colab GPU took 3 minutes, while Jupyter notebook with Apple MPS (like GPU) took 6 minutes. In total, Colab took around 1.5 hours while Jupyter Notebook took 3 hours for 150 epochs.

**(2)** Since our class distribution is perfectly balanced, test and training accuracy alone can be considered for evaluating model performance. According to the "Training and Test Accuracy" Fig 2, the model achieves an accuracy of around 40 on the training set and around 50% on the test set in the beginning. The accuracy gradually increases over the epochs, and towards the end of the training, it reaches a high of around 94% on the training set and around 89% on the test set. A high accuracy on the training set indicates that the model fits the training data well. However, if the validation accuracy is around 5% lower than the training accuracy, it indicates that the model is little overfitting, which means model may not generalize well on new data [3]. Overall, the validation accuracy generally increases with the training accuracy, which is a good that the model is improving. However, there is a point around epoch 30 where the test accuracy seems to start fluctuating a little and starts to become lower training accuracy while training accuracy is not fluctuating like test accuracy. This could indicate that the model is starting to overfit.

**(3)** According to the loss of the "Training and Test loss" Fig 3, the training loss, which measures how well the model fits the training data, reduces gradually from around 2.07 to 1.52 over 150 epochs. On the other hand, the test loss, which measures how well the model is generalizing to new data, also reduces from around 2.00 to 1.57 over 150 epochs but with some fluctuations. After the 100th epoch, the reduction in training loss becomes slower. The validation loss also decreases during the first 50 epochs, indicating that the model is generalizing well on new data. After the 50th epoch, the validation loss stops decreasing and starts to oscillate around a certain value. This is a sign of overfitting [4].

**(4)** From the heatmap, it can be observed that the Dog and Cat images are the most frequently misclassified images, with a misclassification rate ranging from 78% to 83.8%. To improve the model's performance, early stopping could be used to stop training when the validation loss stops improving. Dropout or regularization could also be added to the model to reduce overfitting. Overall, a test accuracy of around 89% is a good performance for image classification [4].

## References:

[1] Ratner, A.J., Ehrenberg, H., Hussain, Z., Dunnmon, J. and Ré, C., 2017. Learning to compose domain-specific transformations for data augmentation. Advances in neural information processing systems, 30.

[2] Chansong, D. and Supratid, S., 2021, March. Impacts of kernel size on different resized images in object recognition based on convolutional neural network. In 2021 9th International Electrical Engineering Congress (iEECON) (pp. 448-451). IEEE.

[3] Aszemi, N.M. and Dominic, P.D.D., 2019. Hyperparameter optimization in convolutional neural network using genetic algorithms. International Journal of Advanced Computer Science and Applications, 10(6).

[4] "How to treat overfitting in CNN" at: https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/