# Project Report

## Project: The Impact of Different Dropout Rate and Batch Normalization on CIFAR-100

# ABOUT PROJECT:

Our project focused on the exploration and optimization of a 3-layer Convolutional Neural Network (CNN) specifically tailored for image classification tasks. The core aim was to deepen our understanding of how varying dropout rates and the implementation of batch normalization affect the performance and efficiency of the CNN.

- **Key Techniques:**
  - **Dropout:** As a crucial regularization technique, dropout was employed at different rates (0.3, 0.5, 0.7) in our model. The primary purpose of dropout is to randomly deactivate a set percentage of neurons during the training process. This randomness helps in preventing the neurons from becoming overly dependent on each other, thus reducing the risk of overfitting. By experimenting with different dropout rates, we aimed to identify the optimal balance between sufficient regularization and maintaining the network's learning capability.

  - **Batch Normalization (BN):** BN normalizes the inputs of each layer to have a mean output activation of zero and a standard deviation of one. This technique enhances the stability and training speed of the neural network and also minimizes the dependency on initialization.

# DATASET SOURCE:

The CIFAR-100 dataset was chosen for this study. This dataset is a collection of 32x32 pixel images categorized into 100 classes, with each class containing 600 images. It is a widely used dataset for benchmarking image classification models.

You can access the CIFAR-100 dataset at:

https://www.cs.toronto.edu/~kriz/cifar.html

# Hardware usage (CPU/GPU/Memory) during training and inferences:

The models for our project were trained using Google Colab. Google Colab provided a powerful and accessible cloud-based platform, enabling us to leverage high-performance hardware without the need for local installation or setup.

**Google Colab Hardware Specifications:**

- **CPU Specifications:**
  - Model: Intel(R) Xeon(R) CPU @ 2.30GHz
  - Number of Cores: 2
  - Threads per Core: 2
  - Clock Speed: 2.30 GHz
  - Architecture: x86_64 (64-bit)
- **GPU Specifications:**
  - Model: NVIDIA Tesla T4
  - Memory: 16 GB
  - CUDA Version: 11.8
  - Driver Version: 525.105.17
- **System Memory**:
  - Total Memory: 12.68 GB

“We conducted a detailed analysis of resource utilization for two specific CNN models: one with a 0.3 dropout rate and batch normalization, and the other with batch normalization but without dropout. Below is a structured comparison of their CPU, memory, and GPU usage.”

## **Model 1:** CNN with 0.3 Dropout and Batch Normalization

- **CPU and Memory Usage**
  - CPU Usage: Fluctuated between 17.8% to 74.1% throughout the training process.
  - Memory Usage: Remained relatively stable, ranging from 32.6% to 33.2%.
- **GPU Usage (NVIDIA Tesla T4)**
  - GPU Temperature: Varied between 50°C and 78°C.
  - Power Usage: Typically ranged from 32W to 43W (out of a 70W capacity).
  - Memory Usage on GPU: Consistently used 1275 MiB out of the total 15360 MiB available.

- GPU Utilization: Ranged from 19% to 31%, indicating active processing time.

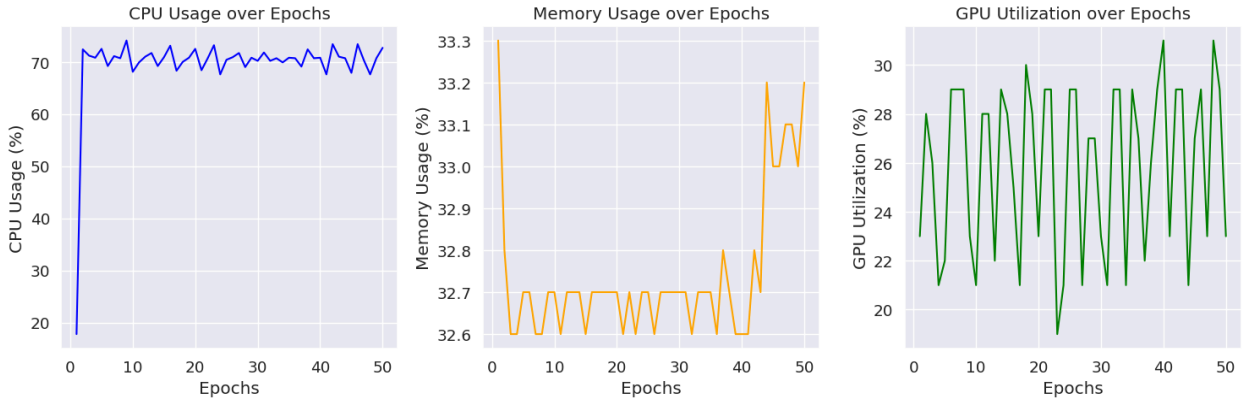


**FIgure:** Resource Utilization Metrics During Computational Epochs

## **Model 2:** CNN Model with Batch Normalization (Without Dropout)

- **CPU and Memory Usage**
  - CPU Usage: Showed significant variation, ranging from 17.8% to 74.1%.
  - Memory Usage: Mostly consistent, staying around 32.6% to 33.2%.
- **GPU Usage (NVIDIA Tesla T4)**
  - GPU Temperature: Fluctuated between 50°C and 78°C.
  - Power Usage: Varied within the range of 32W to 43W.
  - Memory Usage on GPU: Maintained a steady usage of 1275 MiB out of 15360 MiB.
  - GPU Utilization: Ranged approximately from 19% to 31%.

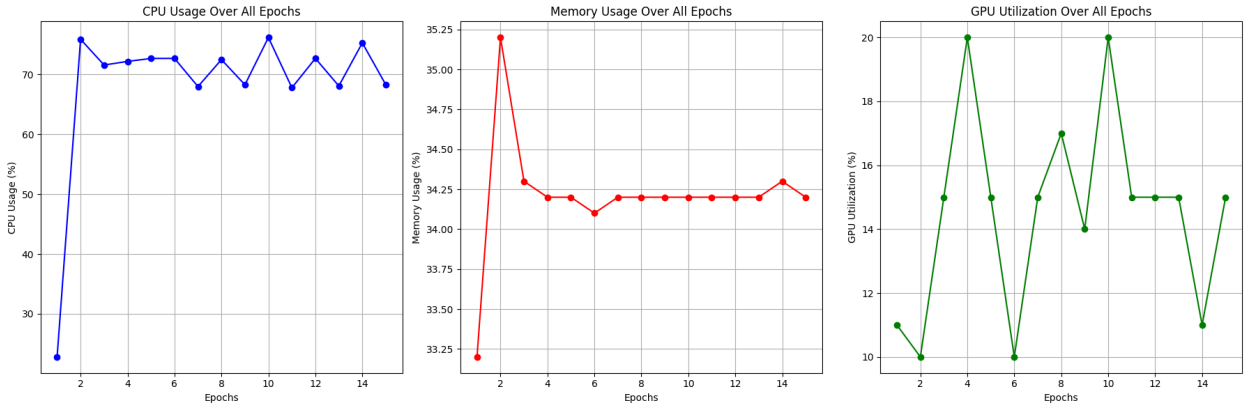


**FIgure:** Resource Utilization Metrics During Computational Epochs

**Note:** Training ceased at epoch 15 with early stopping implemented; the validation loss did not decrease and remained increased for 7 consecutive epochs.

# Models used, Number of Parameters and Number of Operations:

In our study, we trained a series of nine different Convolutional Neural Network (CNN) models. These models were primarily differentiated based on their dropout rates and the inclusion or exclusion of batch normalization. Specifically, the variations in dropout rates were 0.3, 0.5, and 0.7. Each of these configurations was tested under three conditions: with batch normalization, without batch normalization, and without both dropout and batch normalization. This comprehensive approach allowed us to explore the impact of these factors on the model's performance.

For the purpose of detailed analysis, we will focus on two specific models from this series:

1. **CNN Model with a 0.3 Dropout Rate and Batch Normalization:**
   This model incorporates a dropout rate of 0.3 and includes batch normalization. The dropout rate of 0.3 means that during training, 30% of the nodes are randomly omitted in each dropout layer, which helps in preventing overfitting.

2. **CNN Model with Batch Normalization but Without Dropout:**
   This model includes batch normalization but does not employ any dropout layers. Without dropout, all the nodes are retained during training, potentially increasing the model's capacity to learn complex patterns.

## Number of Parameters and Number of Operations:

Since both of the models here have the same layered structure, they have the same number of parameters and operations. Below is the code for it:

In the provided code snippet:

- **Trainable parameters** refer to the elements of the model weights that will be learned from the training data. The model has 5,146,212 such parameters, meaning that many individual data points will be adjusted during the training process to minimize error.

- **FLOPs** stand for floating-point operations, a measure of the computational complexity of the model. The result shows that the model requires approximately 447,528,960 floating-point operations to make a single prediction, which equates to roughly 0.04 billion operations (0.04 GFLOPs). FLOPs, or floating-point operations, typically refer to the combined count of floating-point additions and multiplications required to perform a task.

**For Both CNN models:**

```python
# Calculate and Print Number of Trainable Parameters
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f"The model has {count_parameters(model):,} trainable parameters")

# Calculate and Print FLOPs
from thop import profile
input = torch.randn(1, 3, 32, 32).to(device)
flops, _ = profile(model, inputs=(input, ), verbose=False)
print(f"The model has approximately {flops} FLOPs, which is about {flops / 1e9:.2f} GFLOPs")
```

```
The model has 5,146,212 trainable parameters
The model has approximately 44752896.0 FLOPs, which is about 0.04 GFLOPs
```

```
summary(model, input_size=(3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
         MaxPool2d-3           [-1, 64, 16, 16]               0
            Conv2d-4          [-1, 128, 16, 16]          73,856
       BatchNorm2d-5          [-1, 128, 16, 16]             256
         MaxPool2d-6            [-1, 128, 8, 8]               0
            Conv2d-7            [-1, 256, 8, 8]         295,168
       BatchNorm2d-8            [-1, 256, 8, 8]             512
         MaxPool2d-9            [-1, 256, 4, 4]               0
           Linear-10                 [-1, 1024]       4,195,328
      BatchNorm1d-11                 [-1, 1024]           2,048
          Dropout-12                 [-1, 1024]               0
           Linear-13                  [-1, 512]         524,800
      BatchNorm1d-14                  [-1, 512]           1,024
          Dropout-15                  [-1, 512]               0
           Linear-16                  [-1, 100]          51,300
================================================================
Total params: 5,146,212
Trainable params: 5,146,212
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.00
Params size (MB): 19.63
Estimated Total Size (MB): 21.65
----------------------------------------------------------------
```

I also trained another model to show variation in parameter count and operation since all of my trained models have the same parameters and operation count. Which is VGG-19 below is the number of parameters and operations for that:

**VGG-19**

The model has 20,201,508 trainable parameters, indicating its complexity and potential capacity to learn from data. It requires approximately 0.40 Giga Floating Point Operations (GFLOPs) to make a single forward pass, which gives a sense of the computational effort needed to process an input through the model.

```
[11] # Calculate and Print Number of Trainable Parameters
     def count_parameters(model):
         return sum(p.numel() for p in model.parameters() if p.requires_grad)

     print(f"The model has {count_parameters(model):,} trainable parameters")

     # Calculate and Print FLOPs
     from thop import profile
     input = torch.randn(1, 3, 32, 32).to(device)
     flops, _ = profile(model, inputs=(input, ), verbose=False)
     print(f"The model has approximately {flops} FLOPs, which is about {flops / 1e9:.2f} GFLOPs")
```

```
The model has 20,201,508 trainable parameters
The model has approximately 398307840.0 FLOPs, which is about 0.40 GFLOPs
```

```
# Move the model to the GPU
model = Net().to(device)

#input data is a torch tensor
input_data = torch.randn((1, 3, 32, 32)).to(device)

# Print the model summary to see the number of parameters
summary(model, input_data.shape[1:])
```

```
100%|██████████| 548M/548M [00:05<00:00, 98.1MB/s]
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
              ReLU-2           [-1, 64, 32, 32]               0
            Conv2d-3           [-1, 64, 32, 32]          36,928
              ReLU-4           [-1, 64, 32, 32]               0
         MaxPool2d-5           [-1, 64, 16, 16]               0
            Conv2d-6          [-1, 128, 16, 16]          73,856
              ReLU-7          [-1, 128, 16, 16]               0
            Conv2d-8          [-1, 128, 16, 16]         147,584
              ReLU-9          [-1, 128, 16, 16]               0
        MaxPool2d-10            [-1, 128, 8, 8]               0
           Conv2d-11            [-1, 256, 8, 8]         295,168
             ReLU-12            [-1, 256, 8, 8]               0
           Conv2d-13            [-1, 256, 8, 8]         590,080
             ReLU-14            [-1, 256, 8, 8]               0
           Conv2d-15            [-1, 256, 8, 8]         590,080
             ReLU-16            [-1, 256, 8, 8]               0
           Conv2d-17            [-1, 256, 8, 8]         590,080
             ReLU-18            [-1, 256, 8, 8]               0
        MaxPool2d-19            [-1, 256, 4, 4]               0
           Conv2d-20            [-1, 512, 4, 4]       1,180,160
             ReLU-21            [-1, 512, 4, 4]               0
```

```
              ReLU-18            [-1, 256, 8, 8]               0
         MaxPool2d-19            [-1, 256, 4, 4]               0
            Conv2d-20            [-1, 512, 4, 4]       1,180,160
              ReLU-21            [-1, 512, 4, 4]               0
            Conv2d-22            [-1, 512, 4, 4]       2,359,808
              ReLU-23            [-1, 512, 4, 4]               0
            Conv2d-24            [-1, 512, 4, 4]       2,359,808
              ReLU-25            [-1, 512, 4, 4]               0
            Conv2d-26            [-1, 512, 4, 4]       2,359,808
              ReLU-27            [-1, 512, 4, 4]               0
         MaxPool2d-28            [-1, 512, 2, 2]               0
            Conv2d-29            [-1, 512, 2, 2]       2,359,808
              ReLU-30            [-1, 512, 2, 2]               0
            Conv2d-31            [-1, 512, 2, 2]       2,359,808
              ReLU-32            [-1, 512, 2, 2]               0
            Conv2d-33            [-1, 512, 2, 2]       2,359,808
              ReLU-34            [-1, 512, 2, 2]               0
            Conv2d-35            [-1, 512, 2, 2]       2,359,808
              ReLU-36            [-1, 512, 2, 2]               0
         MaxPool2d-37            [-1, 512, 1, 1]               0
           Flatten-38                  [-1, 512]               0
            Linear-39                  [-1, 256]         131,328
            Linear-40                  [-1, 128]          32,896
              ReLU-41                  [-1, 128]               0
           Dropout-42                  [-1, 128]               0
            Linear-43                  [-1, 100]          12,900
================================================================
Total params: 20,201,508
Trainable params: 20,201,508
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.87
Params size (MB): 77.06
Estimated Total Size (MB): 81.95
----------------------------------------------------------------
```

# Metric

### CNN Model with a 0.3 Dropout Rate and Batch Normalization:

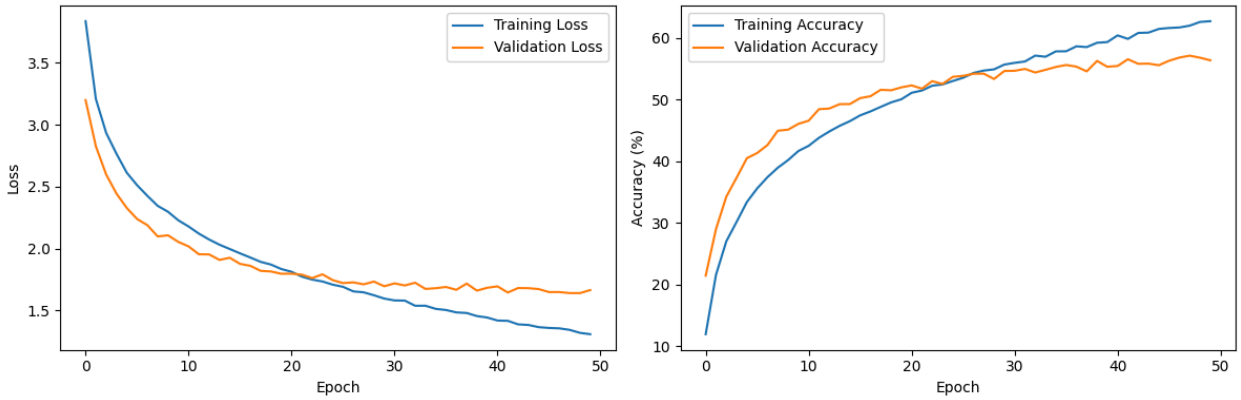


**Fig:** Validation and Accuracy Curve

### CNN Model with Batch Normalization but Without Dropout:

Fi

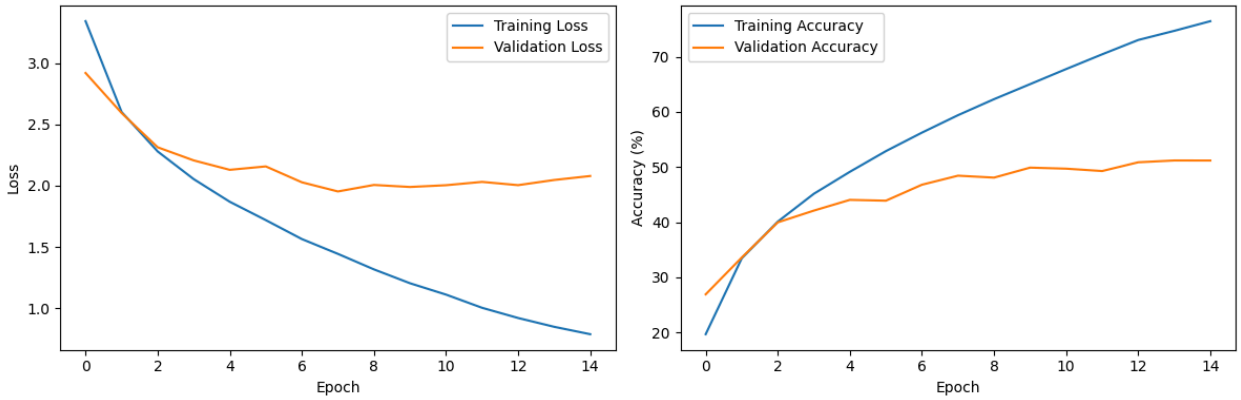


**Fig:** Validation and Accuracy Curve

### VGG19:


**Fig:** Validation and Accuracy Curve

**F1, Precision, Recall, Accuracy Bar plot:**

Since our dataset had 100 classes, the classification report wouldn't be much explanatory. So here's a bar plot instead.