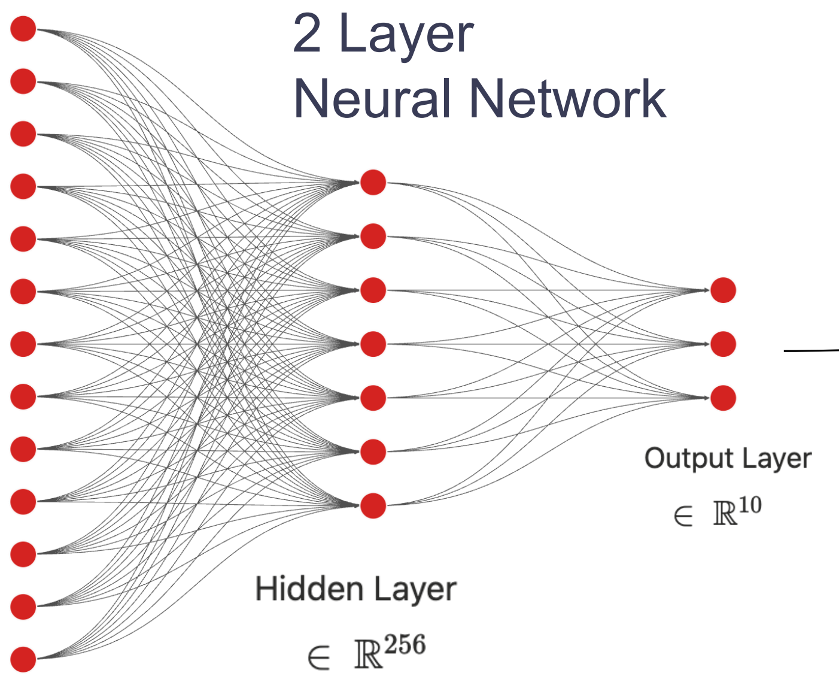
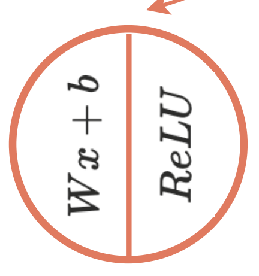
# BatchNorm in Practice - Is it Worth it ?

**Fixed Settings Across each single Dataset:**
Learning rate, Batch size, Epochs, Train/Test size, Loss Type, All other Hyperparameters except optimizer (We will figure out why)

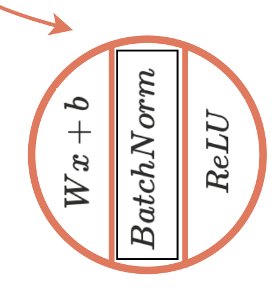
## MNIST Data



```
class NN(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )
```



Neuron without BatchNorm



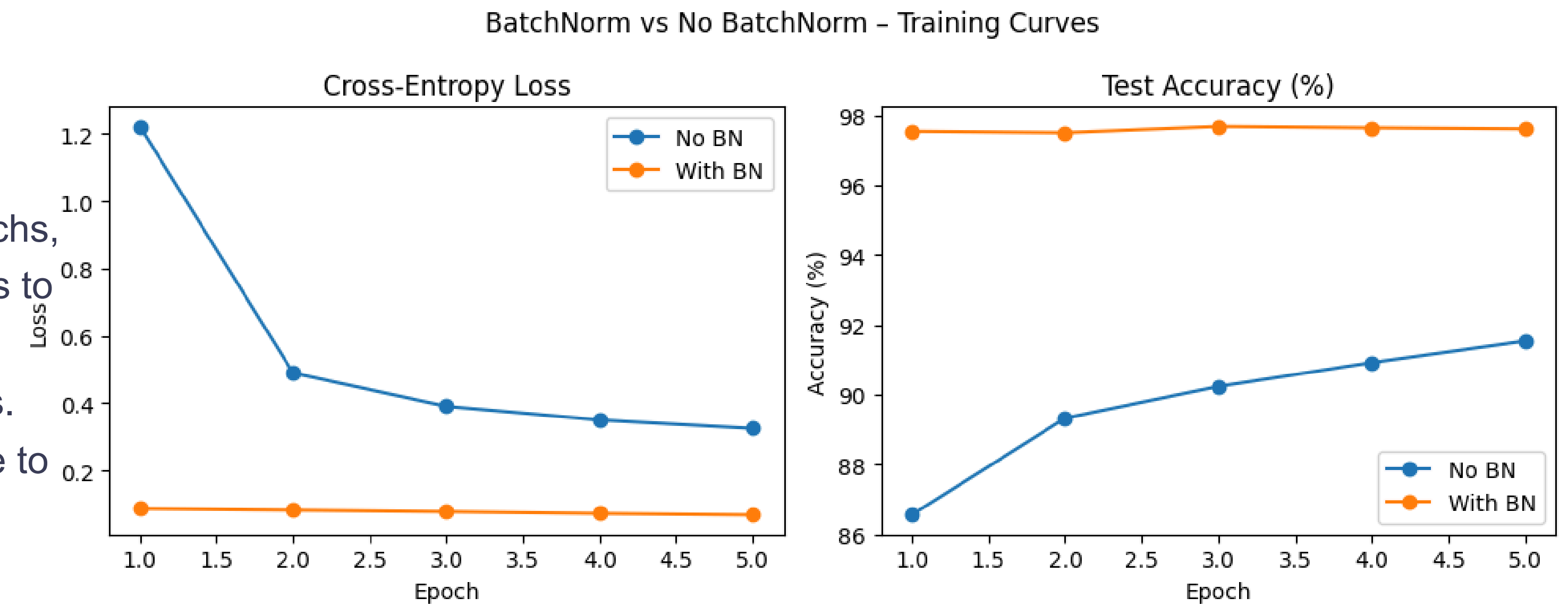
Neuron with BatchNorm
After linear layer and before activation layer

```
class NN_BN(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )
```

### Performance of NN using BatchNorm vs Non BatchNorm using two different Optimizers

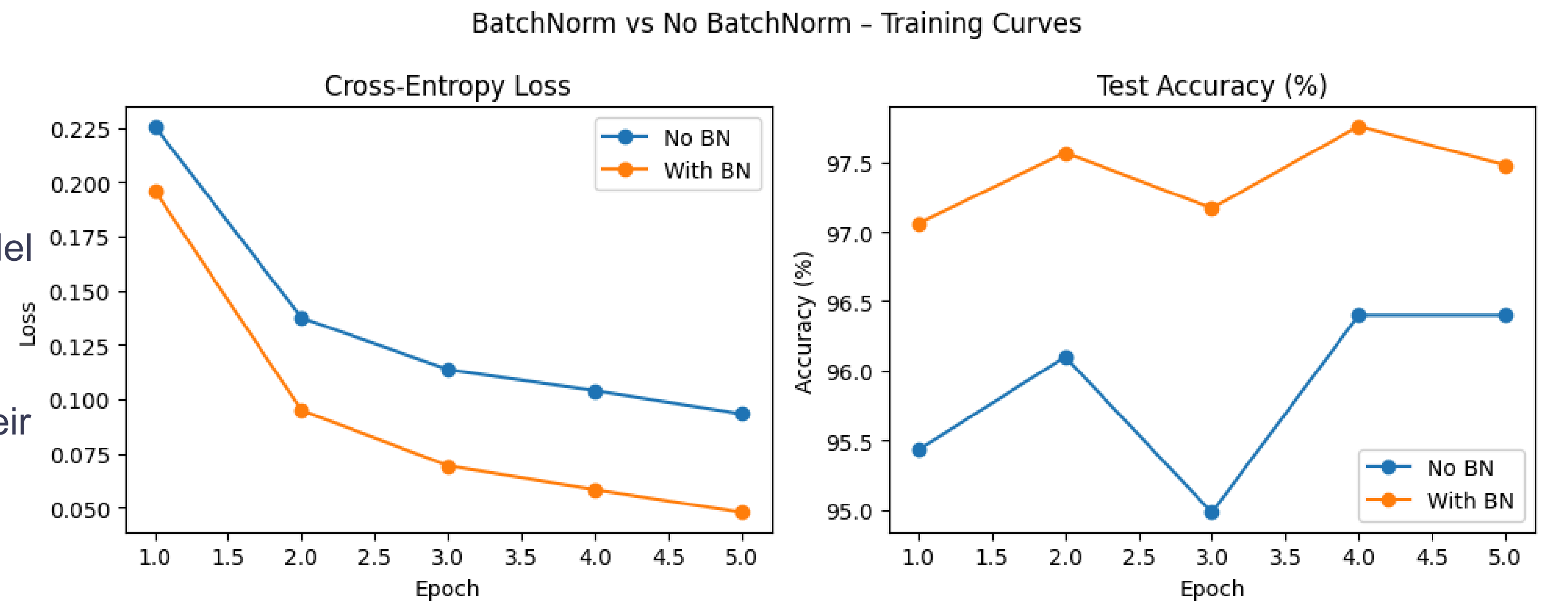
### SGD Optimization

The normalized model consistently starts with a lower loss. Across epochs, the non-normalized model continues to improve, but the batch-normalized model also shows learning progress. This improvement is less visible due to the larger scale of the blue curve values.



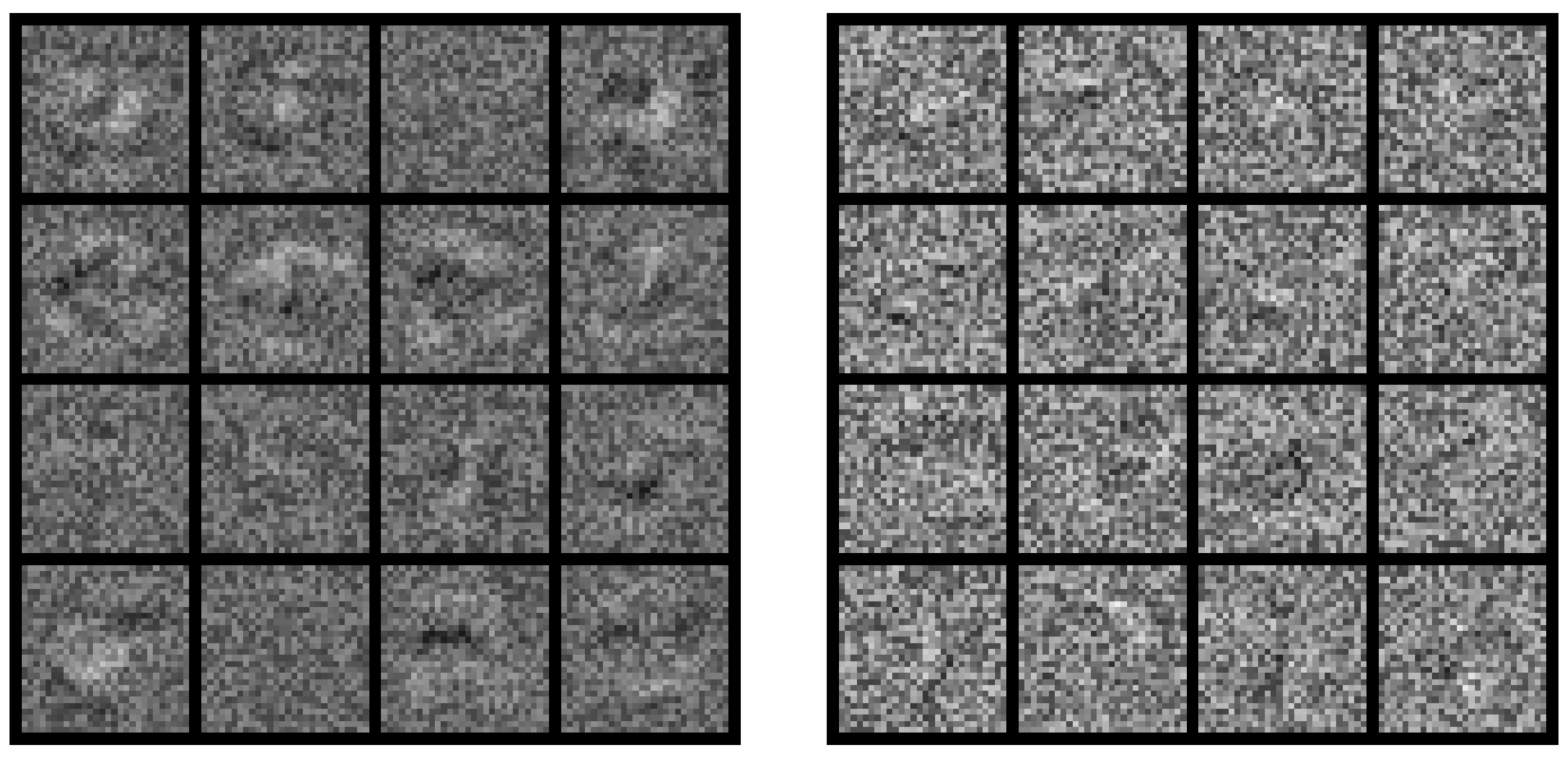
### Adam Optimization

The gap between the two models is noticeably smaller. The faster convergence of the BatchNorm model suggests that as epochs approach infinity, both models may eventually reach similar loss values, though their final accuracies might still differ slightly.



## We show the first linear layer's weights on a 2D grid.

### SGD Optimization
### Non-Batch Normalized VS Batch Normalized



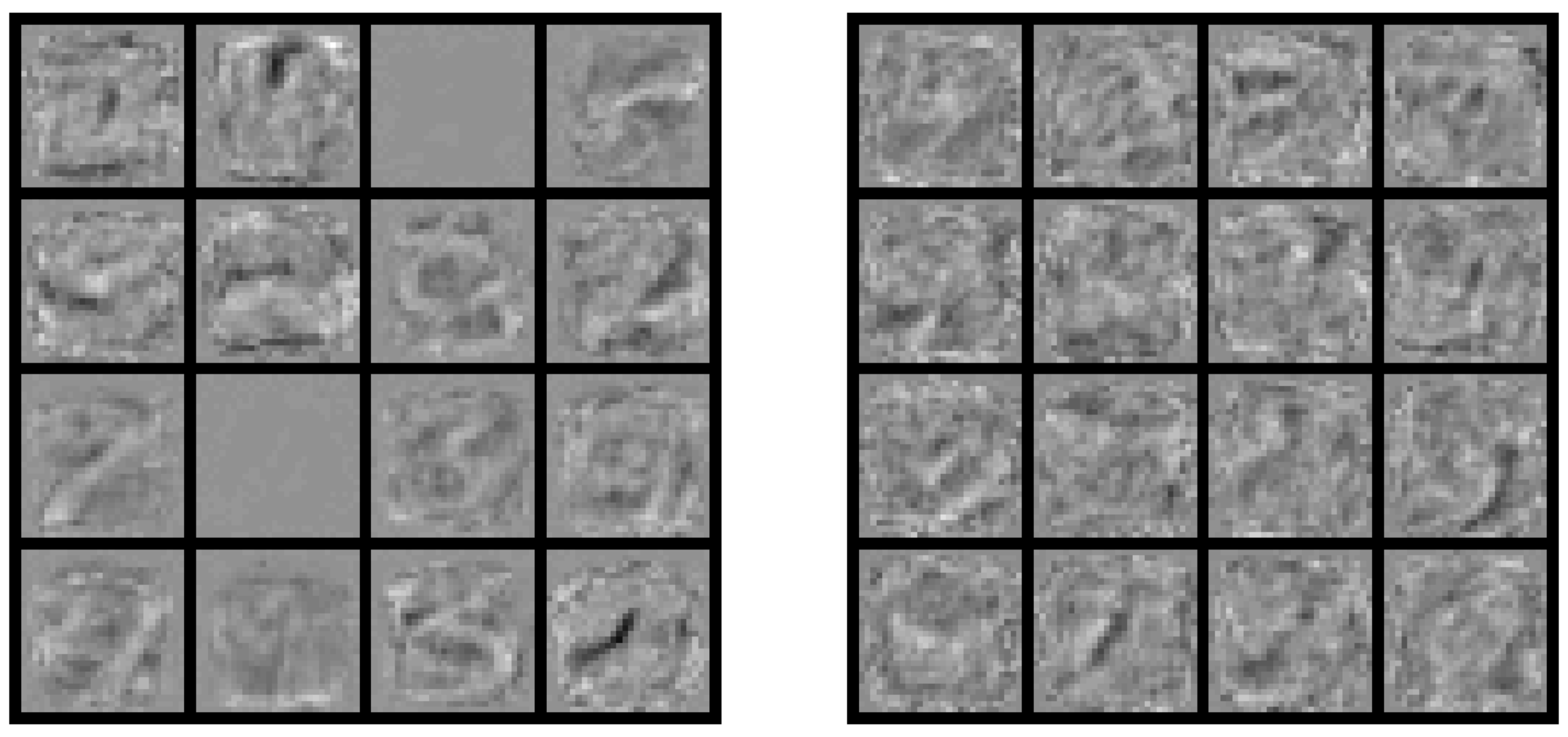
## Results

Regardless of the optimizer, the non-batch-normalized networks tend to produce more interpretable weight patterns. Since these models are trained without normalization, their weights reflect more direct learning.

On the left, Both models still did not reach their optimal minimum, but we can still see tendencies to capture number shapes.

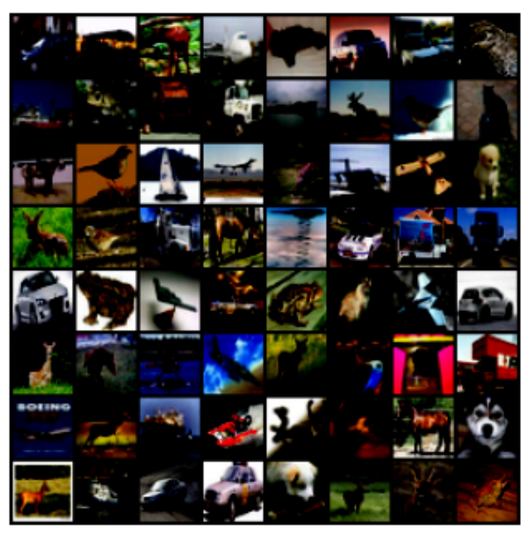
In contrast, on the right in batch-normalized networks trained with Adam, most neurons attempt to learn structured patterns, whereas in non-normalized networks, some neurons exhibit purely noisy activations

### Adam Optimization
### Non-Batch Normalized VS Batch Normalized

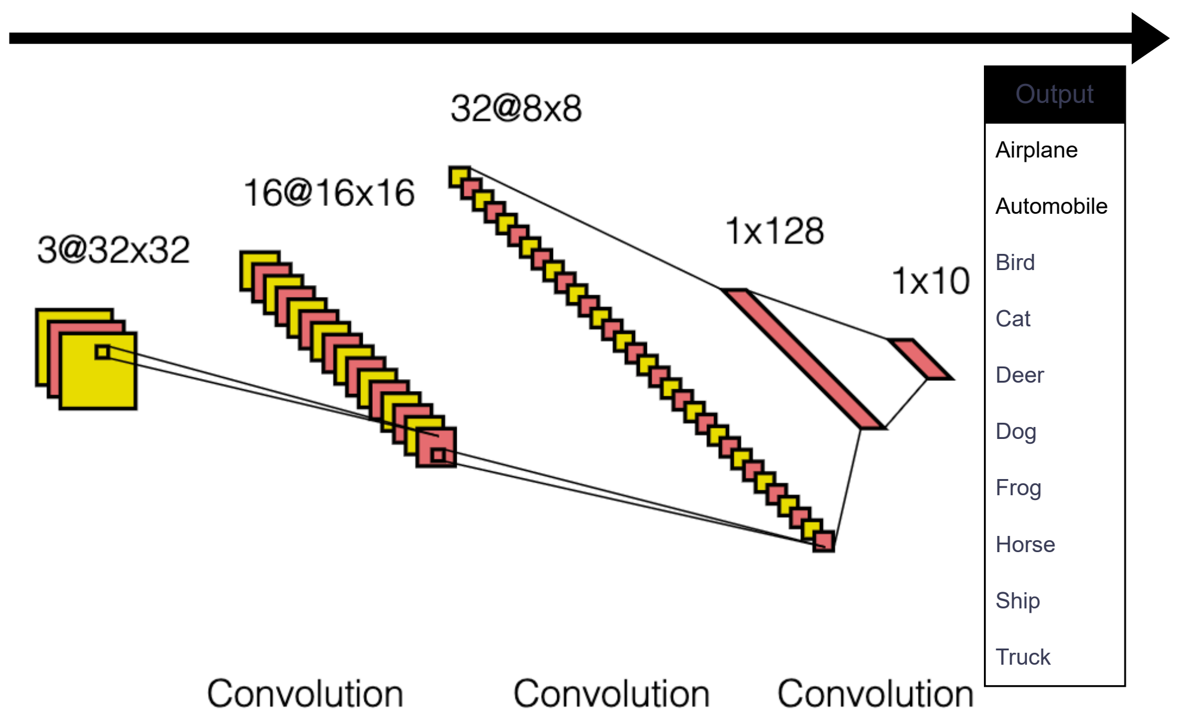


## We proceed with a second experiment of another more complicated dataset

## CIFAR10 Data

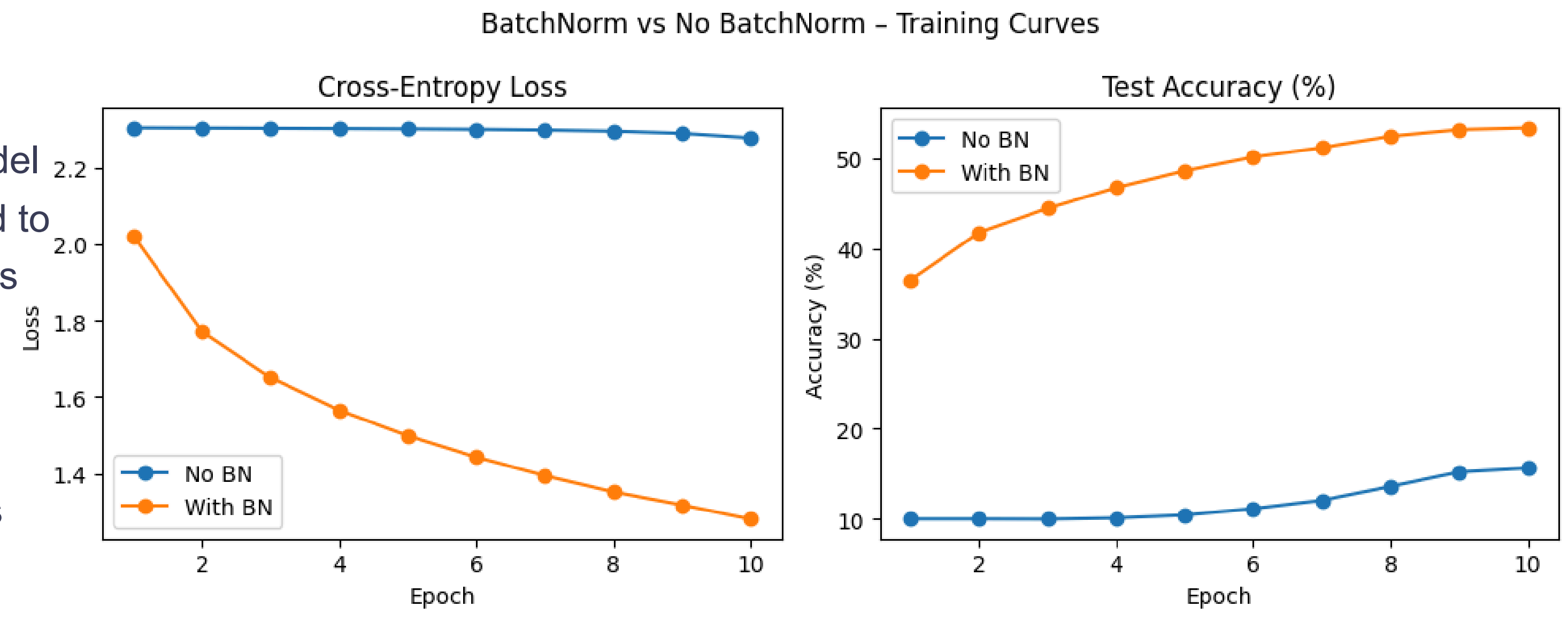


### Performance of NN using BatchNorm vs Non BatchNorm using two different Optimizers



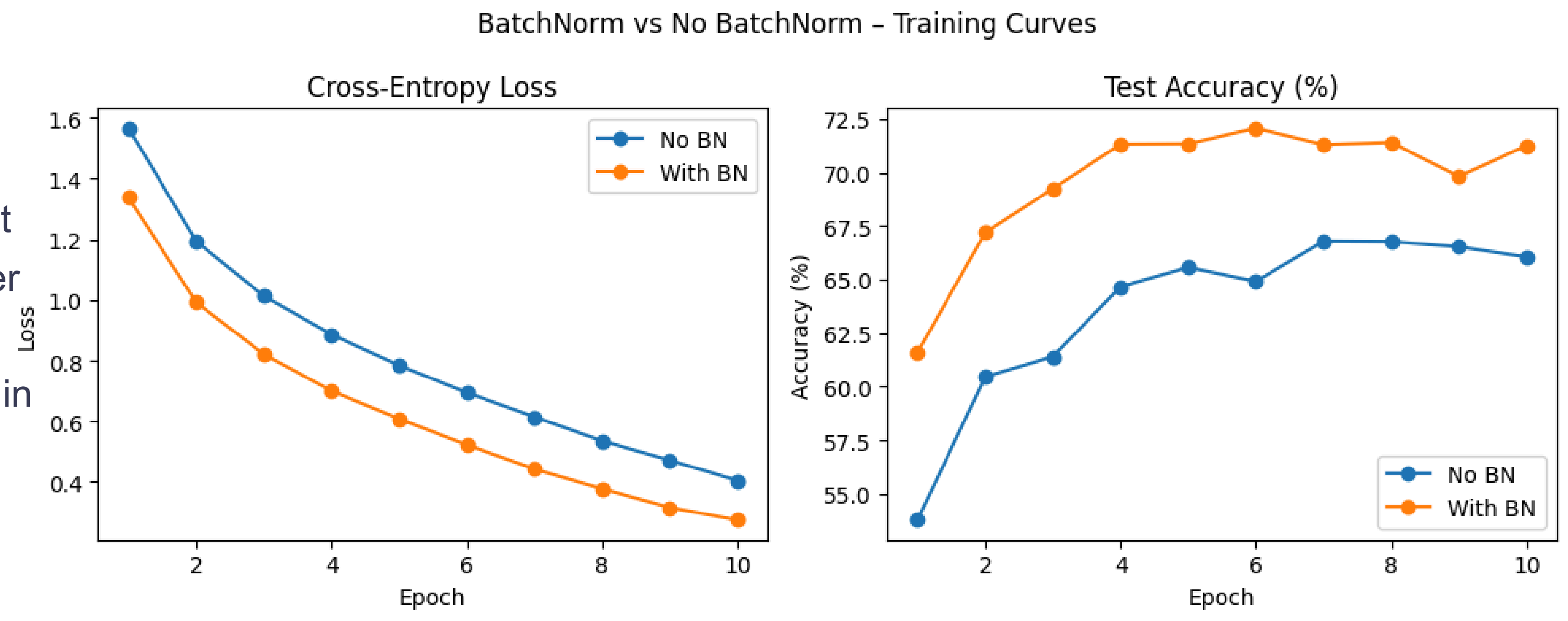
### SGD Optimization

Unlike in MNIST, the normalized model shows faster improvement compared to the other, the growth between epochs is much more considerable when neurons gets normalized. Learning hence is definitely faster. Non-normalized model is worthless in this case.

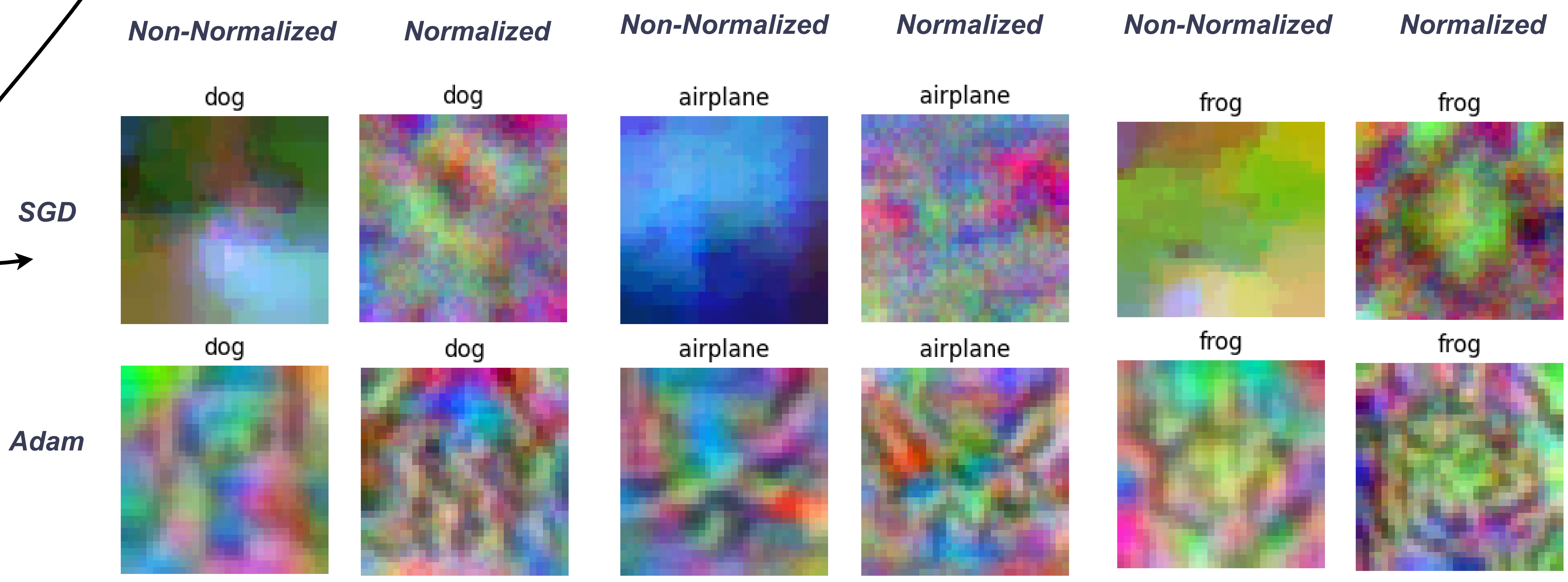


### Adam Optimization

Like in MNIST case, the gap gets smaller between Adam and SGD, but Normalized model always have better accuracy. Again, the goal of using Adam is to showcase what happens in a different optimization setting between models.



**Model hallucination:**
It can be seen as a synthetic image that reflects what the model "believes" the class looks like. We begin with a random noise image, treat its pixels as learnable parameters, feed it through the model, and iteratively adjust it using the model's feedback to approximate the class representation.



## Results

The visualizations show that non-normalized models tend to capture only general details of the class. For example, airplanes appear mostly blue (sky context) and dogs or frogs show greenish tones (natural backgrounds).

In contrast, normalized models generate images with more defined shapes, regardless of whether SGD or Adam optimization is used.

Since in this context Adam enables faster convergence, the batch-normalized model produces the most distinct structures and boundaries, showing the clearest representation of the subject.

In conclusion, for supervised classification tasks, batch normalization generally provides better performance than non-normalized models. While it may reduce some interpretability such as observed in the MNIST experiments, it enhances the model's ability to generate clearer representations.

When computational resources or training time are limited, batch normalization offers a performance boost and even a quick win if you plan to use it in competitions.