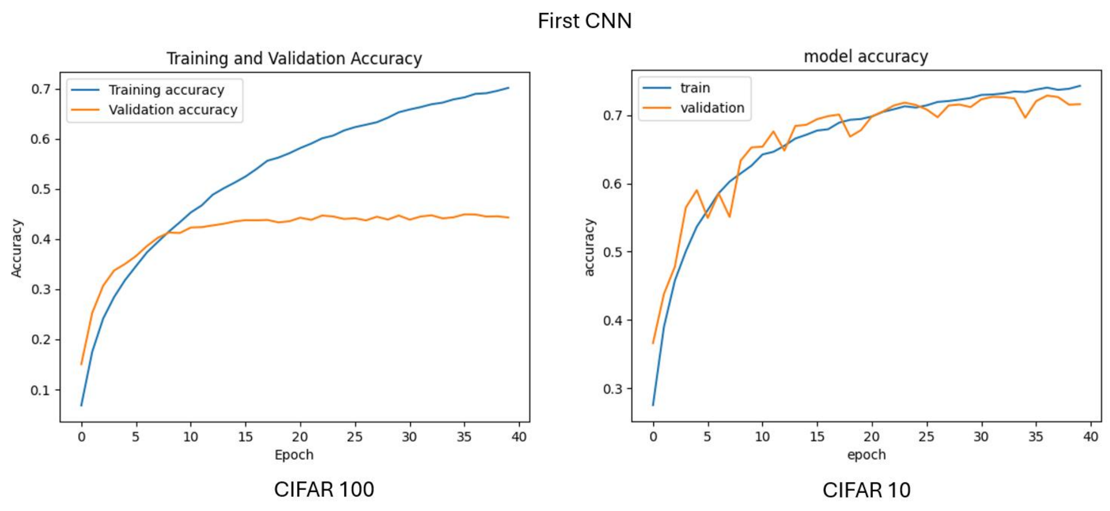
I experimented with different convolution networks on CIFAR 100, which is DNN image classification problem. Like CIFAR 10, this data set contains 50000 train samples and 10000 test samples, but the number of classes increased to 100. This means that for each class, we only have 500 training samples and 100 testing samples, which makes the problem of classifying CIFAR 100 more challenging than classifying CIFAR 10.

The first experiment I did was to use the CNN given from coursework 2, which had a test accuracy of 71% on CIFAR 10. This network contains 12 layers in total:

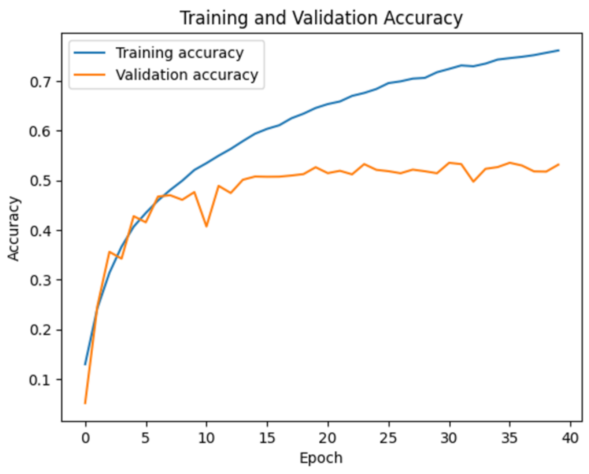
- 4 Convolutional Layers: Extract features from the input images.
- 4 Activation Layers (ReLU): Introduce non-linearity, allowing the model to learn more complex patterns.
- 2 Max Pooling Layers: Reduce the spatial dimensions of the input volume for the subsequent layers, helping to reduce the computation and the number of parameters.
- 3 Dropout Layers: Help prevent overfitting by randomly omitting a subset of features at each iteration.
- 1 Flatten Layer: Converts the 2D feature maps into a 1D feature vector.
- 2 Dense Layers: Fully connected layers that learn non-linear combinations of the high-level features extracted by the convolutional layers, with the last dense layer serving as the classifier.

The figure below shows the training results of using this CNN architecture on CIFAR 10 and CIFAR 100. In both architectures, data are split into 60% training, 20% validation and 20% testing, and both uses 40 eporch. CIFAR 10 has an test accuracy of 71% and CIFAR 100 has an test accuracy of 44%.

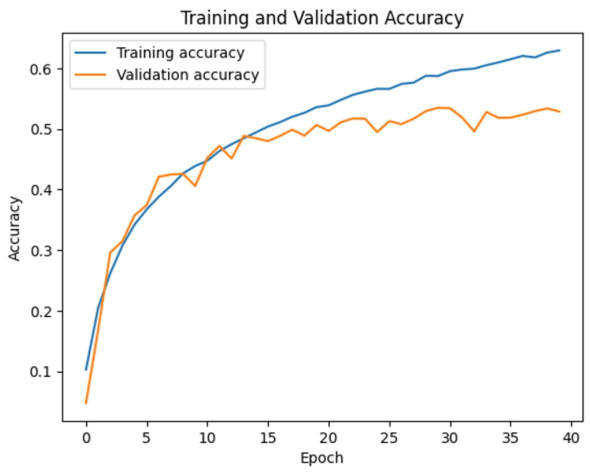


These observations show that while the network performs well on CIFAR 10, there is a lot of over-fitting when training CIFAR 100, causing the validation accuracy to be significantly lower than the training accuracy. This is expected as CIFAR 100 is harder. Next, I tried to modify the network to improve accuracy and reduce overfitting. This is done by increasing the dropout rate and adding batch normalization after each convolution and dense layer. Batch normalization is a technique introduced by Sergey Ioffe and Christian Szegedy (2015). By making normalization for each training mini batch, this technique allows us to use higher learning rates and acts like a regularizer technique, in some cases it can eliminate the need for dropout. The figure below shows the result of the modified network on CIFAR 100, with dropout rate modified:

Added Batch Normalization



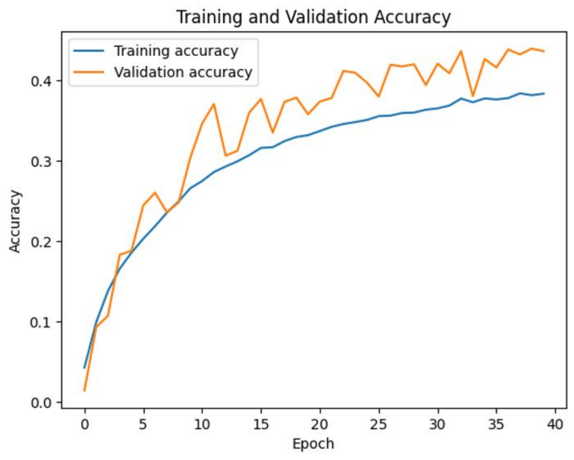
CIFAR 100 – dropout rate = 0.3



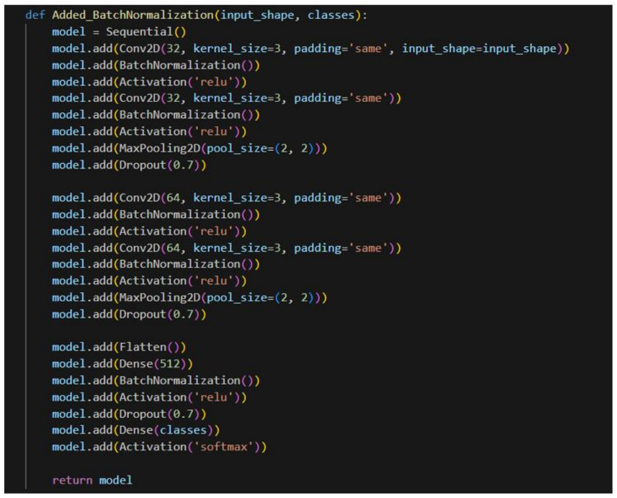
CIFAR 100 – dropout rate = 0.5

Both networks achieved a test accuracy of 54%, which is 10% higher than not using batch normalization. As demonstrated by this figure, further increasing the dropout rate does not increase the test accuracy but decreases the training accuracy so the model does not overfit as much. However, we cannot further increase the dropout rate, as it will affect the model's ability to learn and can cause underfitting. Figures below shows the result of increase the dropout rate to 0.7:

Added Batch Normalization



CIFAR 100 – dropout rate = 0.7



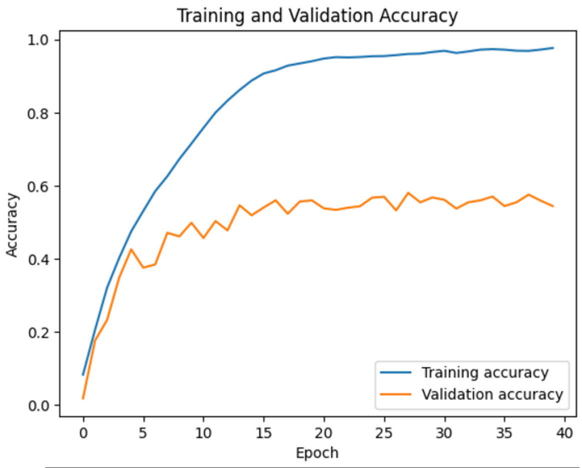
```
def Added_BatchNormalization(input_shape, classes):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=3, padding='same', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(32, kernel_size=3, padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.7))

    model.add(Conv2D(64, kernel_size=3, padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Conv2D(64, kernel_size=3, padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.7))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.7))
    model.add(Dense(classes))
    model.add(Activation('softmax'))

    return model
```

After some trial and error, the best dropout rate for the network would be around 50%, because it closes the gap between training accuracy and validation accuracy without affecting the test accuracy. One interesting observation is that even though increasing the dropout rate to 0.3 – 0.5 decreases overfitting, it does not affect test accuracy. An explanation for this is that this network has reached its generalization capacity, where the model has captured the underlying patterns as well as it can with its given capacity. This explanation seems to fit this situation because the original network was built for CIFAR 10, which is not complex enough to yield the same results on CIFAR 100. We will want to use a more advanced network if we want to increase the test accuracy further.

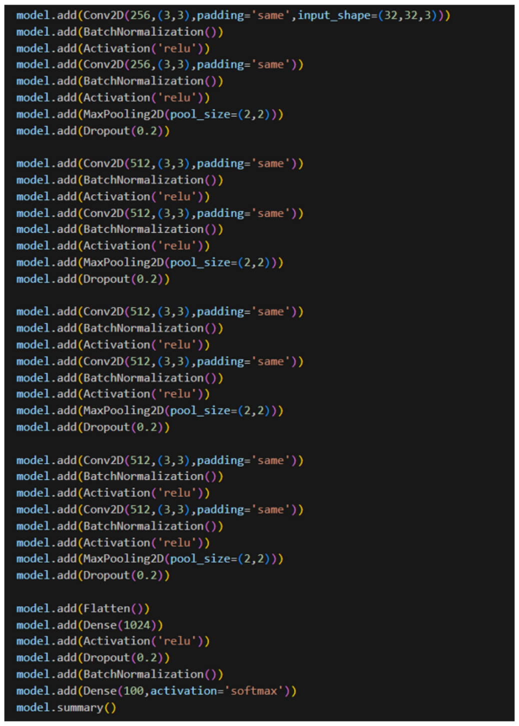
After some research, I found a project that claims to have achieved testing accuracy of 67% only using a simple CNN, which is close to ResNet50's performance on the CIFAR 100 benchmark (L. T. Anh, 2021). They achieved this by increasing the number of filters in convolution layers, the number of convolution layers and using dropout layers, data augmentation and batch normalization.

The original architecture is very large, containing 8 convolution layers with each layer having 256-512 filters, 15795556 parameters, and the network was trained for 350 epochs. Recreating the results of this network is impossible, as it took 7 hours for me to run 40 epochs, the results of these 40 epochs are in the figure below. Because of the limited number of epochs, there are a lot of overfittings in the results. However, this network yields a test accuracy of 55.5%, which already surpassed our network with 54%. This observation shows that we can increase the test accuracy by simply increasing the network size, which we will do for the remainder of this report.

When comparing the modified Lab 2 network and their network, there are several similarities: they both consist of multiple blocks with a sequence of Convolution -> Batch normalization -> ReLU -> Convolution -> Batch normalization -> ReLU -> Pooling ->Dropout. By adding more blocks and adding more filters in each convolution layer, their network performs better, but requires too much training time. I aim to strike a balance by setting the number of filters to between 32-64 and 256-512, which can potentially find a network that can achieve test accuracy higher than 54% and is faster to train.



L.T. Anh's network trained for 40 epochs.

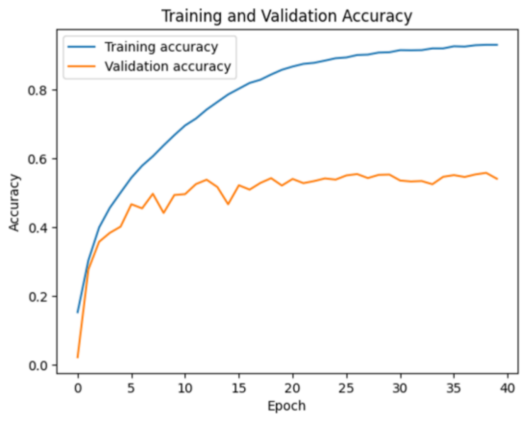


```
model.add(Conv2D(256,(3,3),padding='same',input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(256,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(512,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(100,activation='softmax'))
model.summary()
```

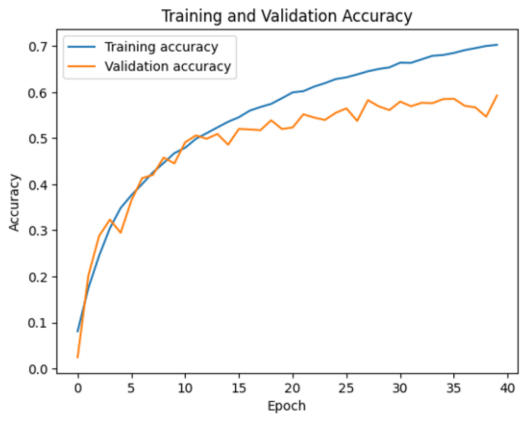
The last network we will go though contains 6 convolution layers, with 64-128 filters in each layer. This network contains 2,762,660 trainable parameters. This network only takes 40 minutes to train 40 epochs and has achieved an accuracy of 59%. By increasing the network size, we can increase the accuracy of the model. However, although L.T.Anh's network is larger than this one, it only reached an accuracy of 55% after 40 epochs. This is because the training environment I used was not suitable for such a large model, hyperparameters such as learning rate need and the number of epoch need to be modified if I want to train L.T.Anh's network efficiently. This observation shows that larger models are not necessarily better models, as the model's performance also depends on the training environment, computational resources, and the task it is trying to solve.

The figure below shows the training result of the last network on CIFAR 100 and CIFAR 10, which have a test accuracy of 59% on CIFAR 100 and 85% on CIFAR 10:

last network, 2.7million parameters on CIFAR 100

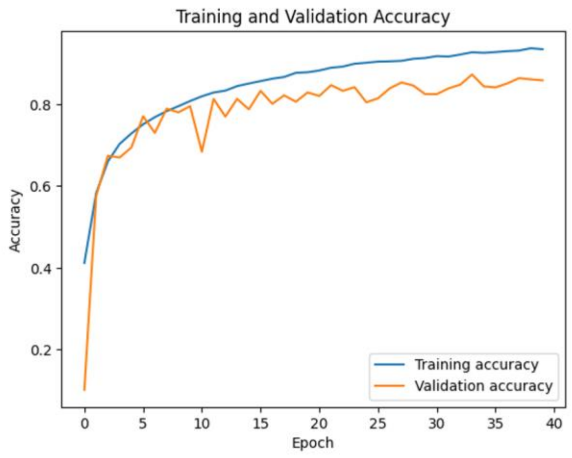


0.2 dropout rate

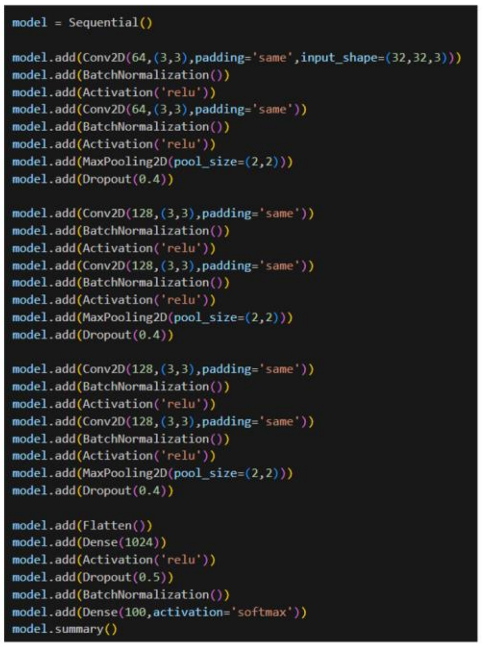


0.4 dropout rate

last network on CIFAR 10 and the network architecture



0.4 dropout rate



```
model = Sequential()

model.add(Conv2D(64,(3,3),padding='same',input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(128,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(128,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(128,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(128,(3,3),padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(100,activation='softmax'))
model.summary()
```

Conclusion:

These experiments demonstrate the deep learning's reliance on data. The model can reach a test accuracy of 85% on CIFAR 10, but only 59% on CIFAR 100. This is because CIFAR 10 has 10 times more data for each class compared to CIFAR 100, and with more data to learn from, the network can perform better. As mentioned in part 1, although deep learning shows promising results on vision task, its data reliance nature of neural networks prevents it from being widely used. Although by modifying the network, such as adding dropout layers and batch normalization, we can help the network perform better on existing networks, the underlying problem still exists. Future research needs to make models more resilient to new scenarios without the need of extensive retraining on large datasets.

References:

L. T. Anh, 2021. *CNN-CIFAR-100: A simple CNN model in Keras without transfer learning achieves 67% accuracy on test set of CIFAR-100* [Online]. Available at: <https://github.com/LeoTungAnh/CNN-CIFAR-100> [Accessed 15 April 2024].

Ioffe, S. and Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167v3*.