Intel Image Classification using Convolutional Neural Networks

- **Artemiy Yalovenko - CSC 578-910**
- Kaggle**: artemiyyalovenko**
- Kaggle Leaderboard place: **12**

Overview of Baseline model

Our baseline was a basic CNN model designed to classify images into six categories. The base model had Conv2D, MaxPooling2D, Flatten, Dense, and Dropout layers. It utilized 64 filters in the convolutional layer, a kernel size of 3x3, a dropout rate of 25 percent, and a dense layer of 128 units. The model showed poor performance as there was a significant gap between training and validation accuracy and loss. Thus, the goal was to fit this overfitting which was especially evident given the fact that loss was increasing on validation set with each epoch (image 1).

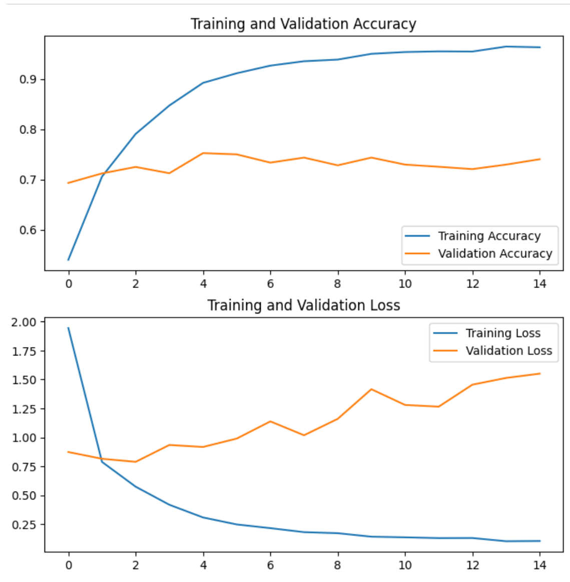


image 1

Model parameter tuning process

The first major adjustment was the addition of a LR scheduler. I was hoping to address validation loss plateaus by reducing the learning rate by half after 3 epochs of non-improvement. It also allowed me to focus on tuning other hyperparameters for the most part. While this change improved training performance, validation metrics plateaued early and LR change didn't help. However it slightly made the model more stable (image 2).

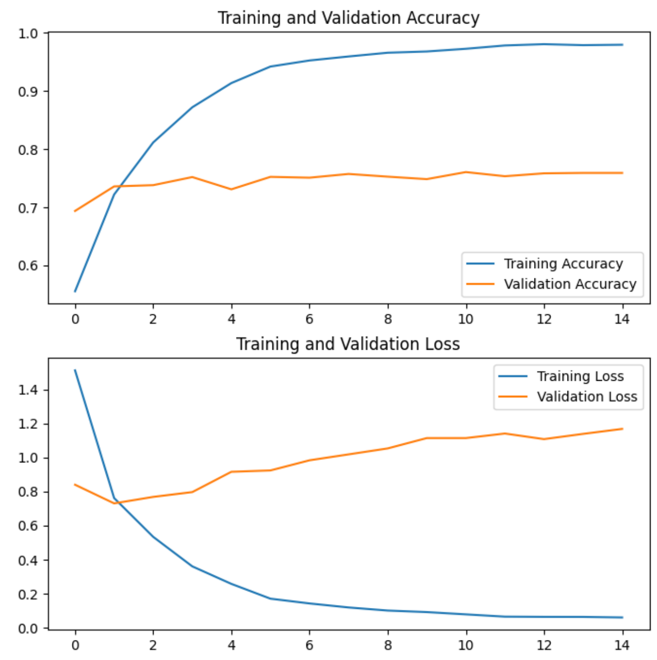


image 2

Then I added L2 regularization was added to generalize the model's learning. It resulted in overall decrease in accuracies, but improved overfitting a lot. There was a lot of volatility in the validation accuracy which also was relatively stagnant and validation loss plateaued early. Still the decrease in overfitting led me to believe that I needed to adjust the regularization penalty. (image 3) Note though, that in another run, the model has more expected behavior when accuracies rising and loss falling (image 3.2).

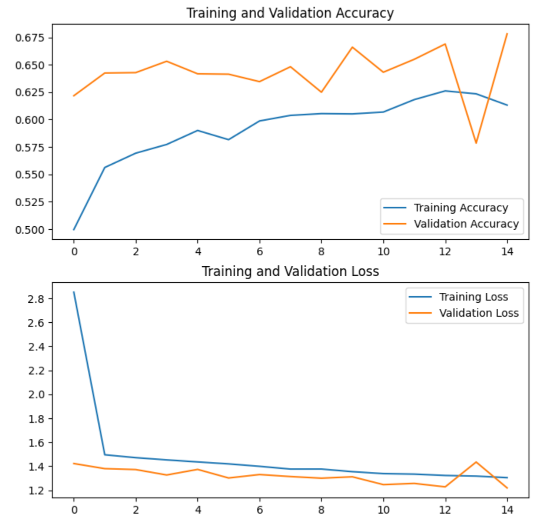


image 3

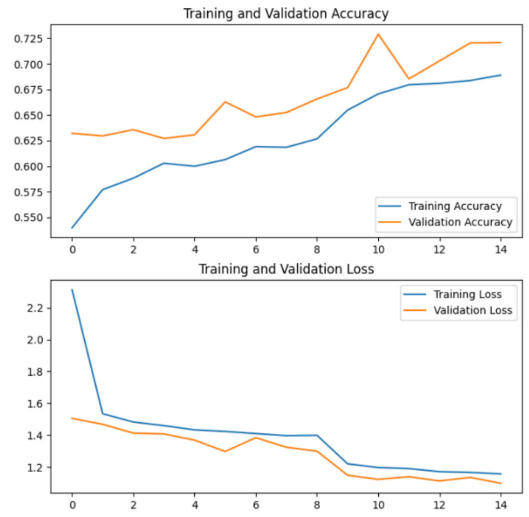


image 3.2

The L2 regularization was reduced to 0.001 but the overfitting seemed to have been efficiently dealt with only in the early epochs (image 4). In response, regularization strength was increased to 0.0025, and an additional dropout layer was introduced, which helped align training and validation accuracies more closely, as demonstrated in Image5.

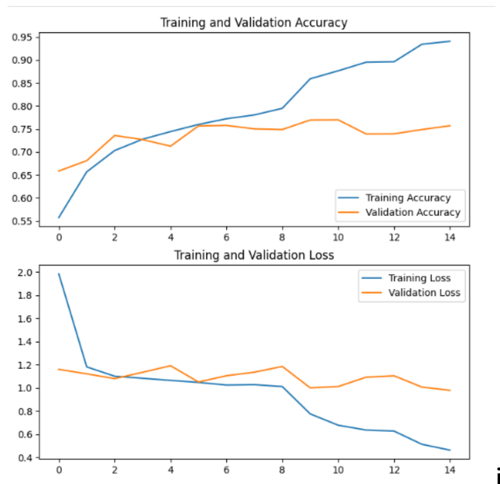


image 4

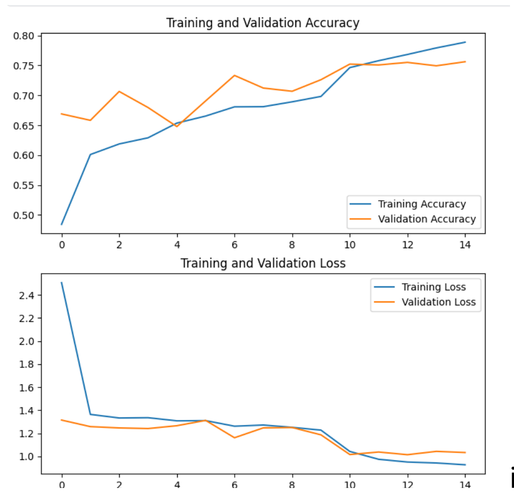


image 5 (aka model5)

Next, I attempted to add Batch Normalization into the model's architecture. I needed to normalize the inputs of each layer, to improve model stability which was still a bit choppy. I also thought it could reduce overfitting. The result was poorer than the prior model which led me to believe the model had somehow increased sensitivity to specific features or batches in the validation data. I now assume it may have been because it was combined with L2 regularization, but at the time it didn’t hit me. There were also lots of overfitting and lots of variance across validation accuracy and loss metrics. (image 6)

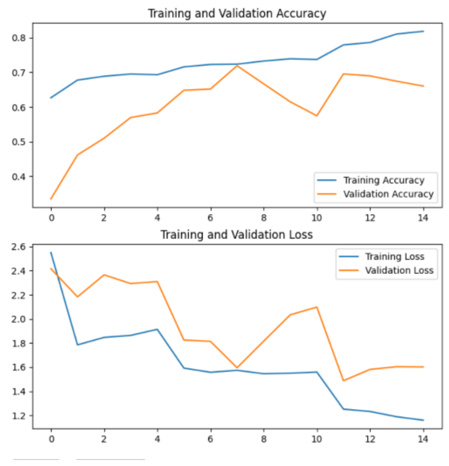


image 6

I attempted to address the overfitting seen earlier by increasing the dropout rates after each convolutional and dense layer. The combination of enhanced dropout, Batch Normalization, and L2 regularization was expected to ensure the model's capability to generalize. It kind of did that but the accuracies among train and validation, while high (90% and 80% respectively) were too far apart and the validation acc and loss metrics were still extremely unstable as seen by wild fluctuations in image 7.

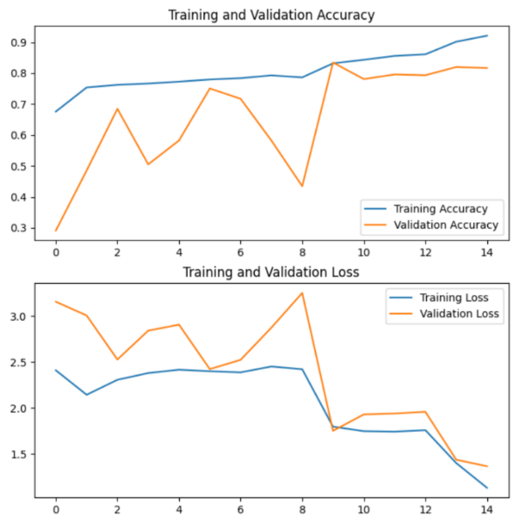


image 7

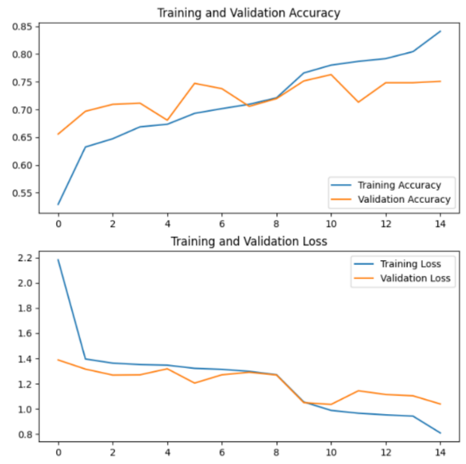
I thought I may have overstepped and decided to return to a simpler model with model5, which lacks batch normalization and has 1 less Convolutional Block. Taking slow, I focused on adjusting dropout rates one by one to improve regularization. I needed to see where i can improve generalization with the simplest solution.

(Image 8) showed that increasing dropout after convolutional layers stabilized training accuracy around 85%, which is considered high. But validation accuracy fluctuated through the epochs indicating instability. The val_acc peaked at 75% and then declined, thus there was still overfitting.

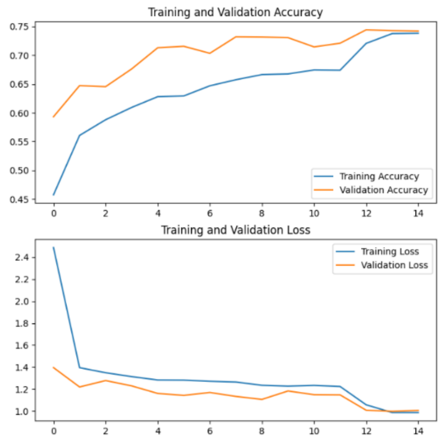
(Image 9) tested increased dropout before the output layer, leading to steady training accuracy and improved validation metrics. Generalization and model stability was better given the decrease in validation loss.

(Image 10) combined increased dropout rates at multiple layers yielded very similar results to the second setup but with slightly worse across loss metrics. The combination did not significantly improve accuracy beyond the improvements seen in the second setup and seemingly decreased stability.

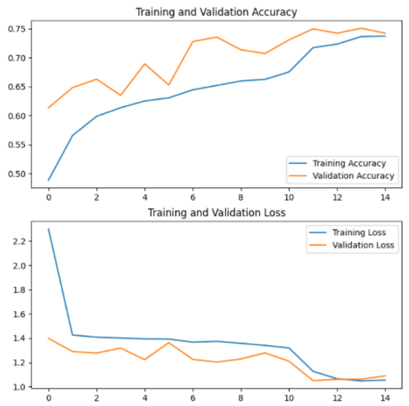
I concluded that increasing dropout before the output layer provided the best balance, enhancing both learning capacity and generalization. This approach was kept going into the next models where I attempted to adjust models to capture more complexity now that the overfitting was seemingly handled.



image(8)



(image 9)



(image 10)

To capture more complexity, I experimented with filter sizes and adjusted the convolutional layer architecture. Using the model with increased dropout before the output layer (model5_2), I modified the first layer with 32 (3x3) filters for fine details, a second layer of 64 (5x5) filters designed to capture more broadly, and added a third layer with 128 (3x3) filters.

This model tested with noticeable improvements in the model's learning trajectory (image 11). Training accuracy went from 43% to 78% with barely any volatility across epochs. Validation accuracy started at 61% and peaked around 80.5% but ended up at 79.7%. This was the highest accuracy I've gotten so far and the loss metrics validated the stability of the model while also ending up at the lowest loss values in my process so far. Model's generalization ability was strong and it was performing admirably.

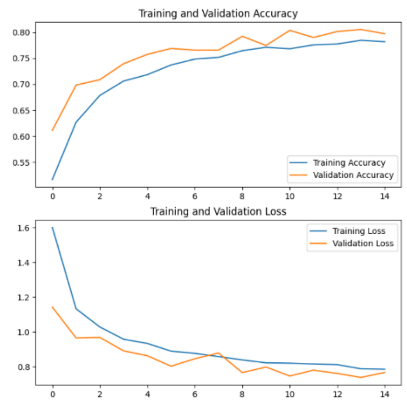


image 11 (results of modified model5_2)

Best model overview

Then a better model was found adjusting eta parameter to 0.0001 and regularization parameter set to 0.001 instead of 0.0025. The LR scheduler wasn't activated with the new eta, and the model seemingly captured more complexity and was more stable. The model trajectory across metrics can be seen in image 12. The final validation accuracy was 1% higher than the training during fitting and the validation loss was 2 % lower (seemingly not shown in the image 12 charts, but can be seen in image 12.2) potentially indicating that with more epochs there is potential of better model performance. Overall the model5_x performed 1-2% higher in validation accuracy and loss than the modified model5_2 which is why it was chosen as the best model.

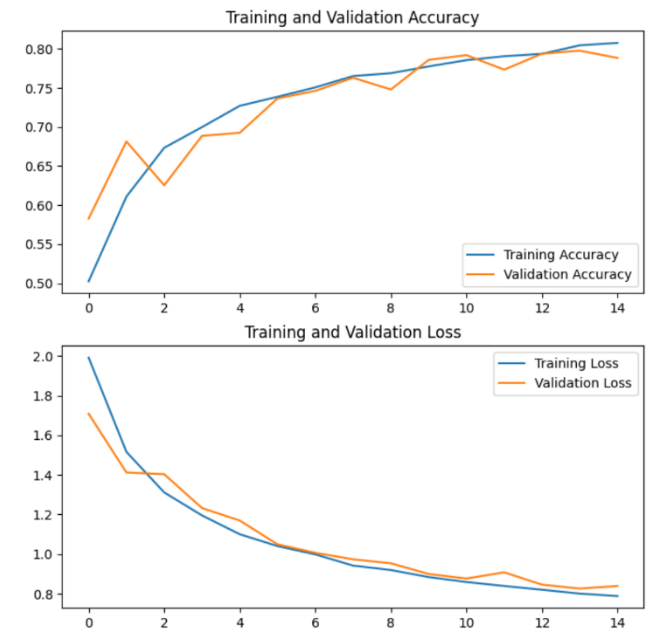
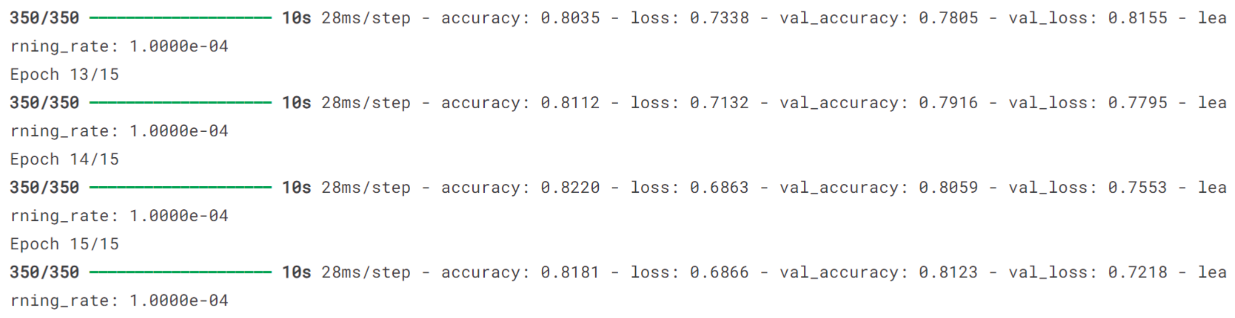


image 12



```
350/350 ━━━━━━━━━━━━━━━━━━━━ 10s 28ms/step - accuracy: 0.8035 - loss: 0.7338 - val_accuracy: 0.7805 - val_loss: 0.8155 - lea
rning_rate: 1.0000e-04
Epoch 13/15
350/350 ━━━━━━━━━━━━━━━━━━━━ 10s 28ms/step - accuracy: 0.8112 - loss: 0.7132 - val_accuracy: 0.7916 - val_loss: 0.7795 - lea
rning_rate: 1.0000e-04
Epoch 14/15
350/350 ━━━━━━━━━━━━━━━━━━━━ 10s 28ms/step - accuracy: 0.8220 - loss: 0.6863 - val_accuracy: 0.8059 - val_loss: 0.7553 - lea
rning_rate: 1.0000e-04
Epoch 15/15
350/350 ━━━━━━━━━━━━━━━━━━━━ 10s 28ms/step - accuracy: 0.8181 - loss: 0.6866 - val_accuracy: 0.8123 - val_loss: 0.7218 - lea
rning_rate: 1.0000e-04
```

image 12.2