Week 3 Homework

By Vijay Venkatesan

MNIST Notebook: Link here

CIFAR10 Notebook: Link here

Cats, Dogs, and Wolves Notebook: Link here

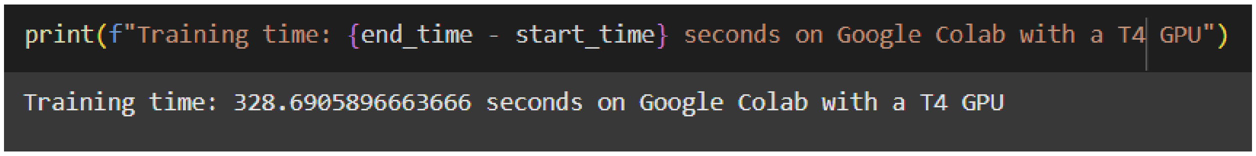
10 Big Cats of the Wild Notebook: Link here

10 Big Cats of the Wild Dataset: Link here

Task 1

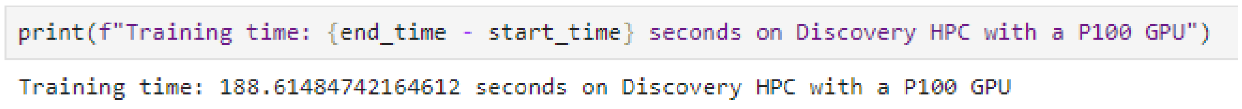
MNIST Dataset

Google Colab



```
print(f"Training time: {end_time - start_time} seconds on Google Colab with a T4 GPU")

Training time: 328.6905896663666 seconds on Google Colab with a T4 GPU
```

Discovery HPC



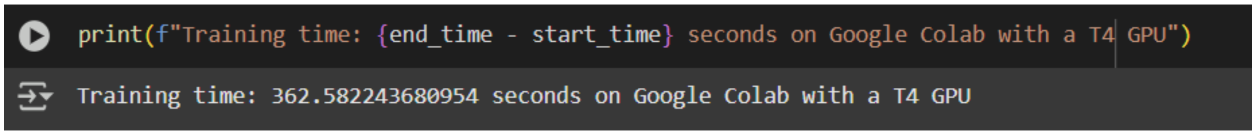
```
print(f"Training time: {end_time - start_time} seconds on Discovery HPC with a P100 GPU")

Training time: 188.61484742164612 seconds on Discovery HPC with a P100 GPU
```

As shown above, training a CNN on the MNIST dataset on Google Colab took about 330 seconds with a T4 GPU. On the other hand, training on Discovery HPC took about 190 seconds with a P100 GPU. The P100 GPU on Discovery HPC was approximately 43% faster than the T4 GPU on Google Colab. This may be attributed to the fact that the P100 GPU with 3584 CUDA cores is greater compared to the T4 GPU with 2560 CUDA cores. Additionally, the P100 GPU has a stronger memory bandwidth of 732 GB/s whereas the T4 GPU has less than half this capacity with only 320 GB/s.

The MNIST dataset consists of grayscale images of handwritten digits of size 28 x 28 pixels. There are 10 classes in this dataset, corresponding to the digits 0 to 9. The MNIST dataset is structured as 60,000 images in the training set and 10,000 images in the test set. The TensorFlow library was used to load the dataset. The parameters that were set up included deciding how many convolution and max pooling pairs to included in the CNN architecture. Additionally, deciding the number of filters to use in each convolutional layer to help with feature extraction.

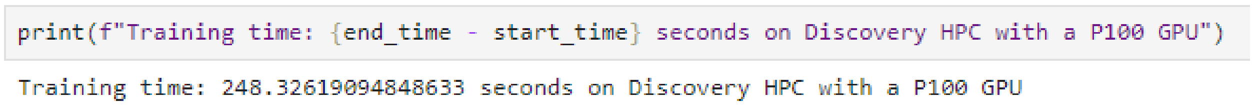
CIFAR10 Dataset

Google Colab



```
print(f"Training time: {end_time - start_time} seconds on Google Colab with a T4 GPU")
Training time: 362.582243680954 seconds on Google Colab with a T4 GPU
```

Discovery HPC



```
print(f"Training time: {end_time - start_time} seconds on Discovery HPC with a P100 GPU")
Training time: 248.32619094848633 seconds on Discovery HPC with a P100 GPU
```

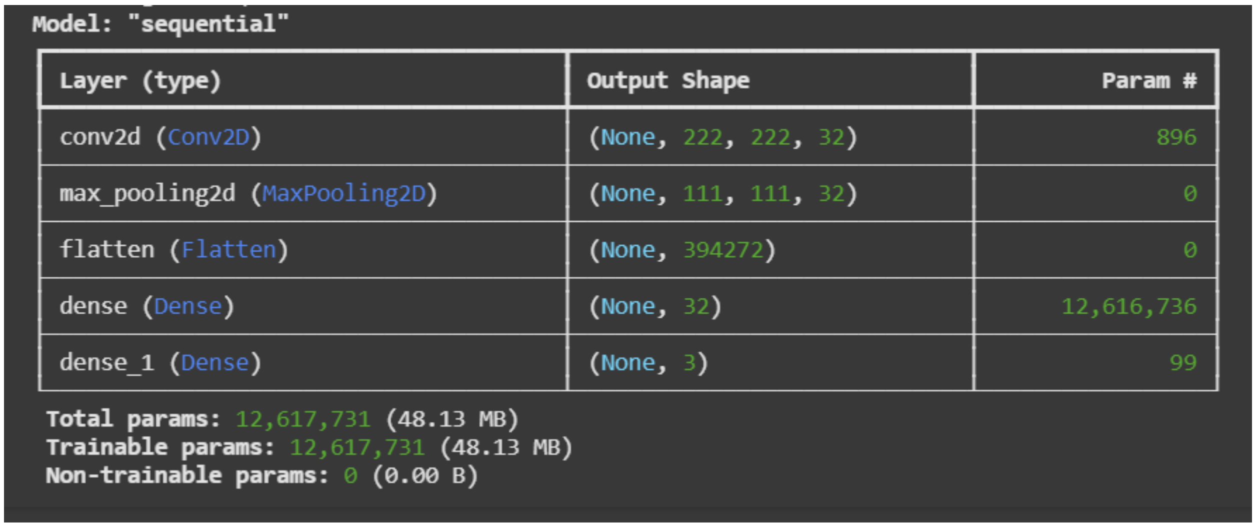
As shown above, training a CNN on the CIFAR10 dataset on Google Colab took about 360 seconds with a T4 GPU. On the other hand, training on Discovery HPC took about 250 seconds with a P100 GPU. The P100 GPU on Discovery HPC was approximately 32% faster than the T4 GPU on Google Colab. The performance difference may be attributed to similar reasons as stated above.

The CIFAR10 dataset consists of color images, RGB values, of 10 object categories. The size of each image is 32 x 32 pixels. There are 10 classes in the dataset, each for one of the types of objects. The objects were diverse. Some of the object types included airplanes, birds, frogs, ships, and trucks. The CIFAR10 dataset consists of 50,000 images in the training set and 10,000 images in the test set. The TensorFlow library was used to load the dataset. The parameters that were set up included deciding how many convolution and max pooling pairs to include in the CNN architecture. Additionally, deciding the number of filters to use in each convolutional layer to help with feature extraction. Since the CIFAR10 dataset is more complex than the MNIST dataset, the number of filters in each subsequent convolutional layer increased to help capture more complex features at deeper layers of the CNN.

Task 2

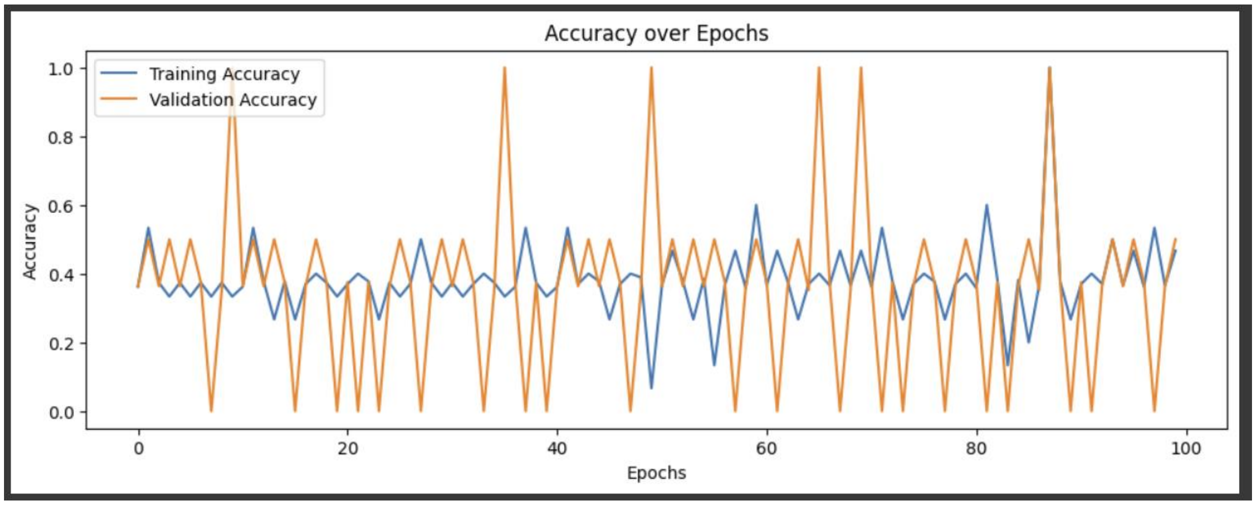
Experimentation with various CNN architectures [1 to 3 Convolution/Max Pooling Pairs]

1 Convolution/Max Pooling Pair

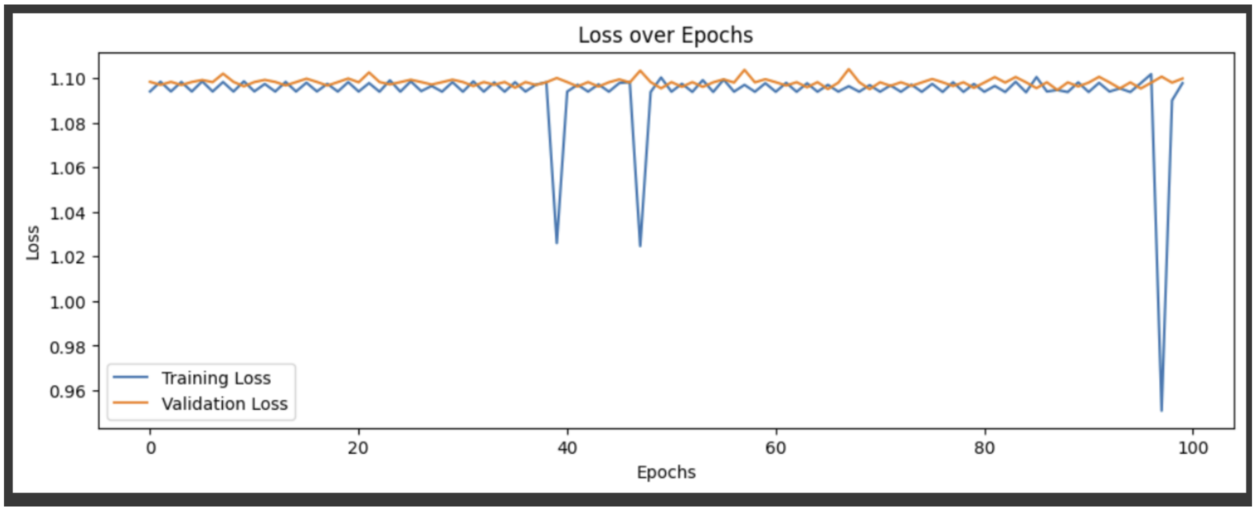


```
Model: "sequential"
┌─────────────────────────────┬──────────────────────────┬─────────────┐
│ Layer (type)                │ Output Shape             │     Param # │
├─────────────────────────────┼──────────────────────────┼─────────────┤
│ conv2d (Conv2D)             │ (None, 222, 222, 32)     │         896 │
├─────────────────────────────┼──────────────────────────┼─────────────┤
│ max_pooling2d (MaxPooling2D)│ (None, 111, 111, 32)     │           0 │
├─────────────────────────────┼──────────────────────────┼─────────────┤
│ flatten (Flatten)           │ (None, 394272)           │           0 │
├─────────────────────────────┼──────────────────────────┼─────────────┤
│ dense (Dense)               │ (None, 32)               │  12,616,736 │
├─────────────────────────────┼──────────────────────────┼─────────────┤
│ dense_1 (Dense)             │ (None, 3)                │          99 │
└─────────────────────────────┴──────────────────────────┴─────────────┘
 Total params: 12,617,731 (48.13 MB)
 Trainable params: 12,617,731 (48.13 MB)
 Non-trainable params: 0 (0.00 B)
```

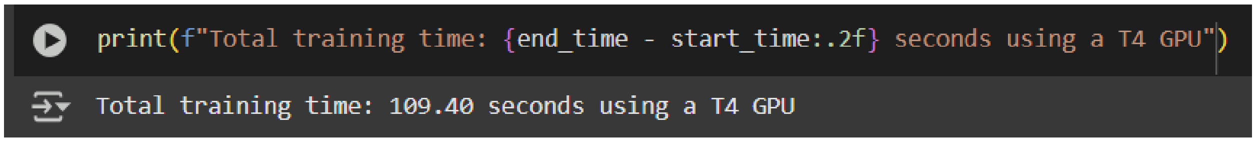
The above model summary indicates that 1 convolution/max pooling pair is included in the CNN architecture.



The above screenshot plots the number of epochs on the x-axis and the accuracy on the y-axis. As shown above, the training accuracy fluctuates between 0.4 and 0.6. The accuracy on the validation curve follows a similar pattern. Both the training curve and validation curve have many jagged points indicating the model may need further training. To combat this instability in learning, tinkering with hyperparameters such as the learning rate or batch size may be needed.



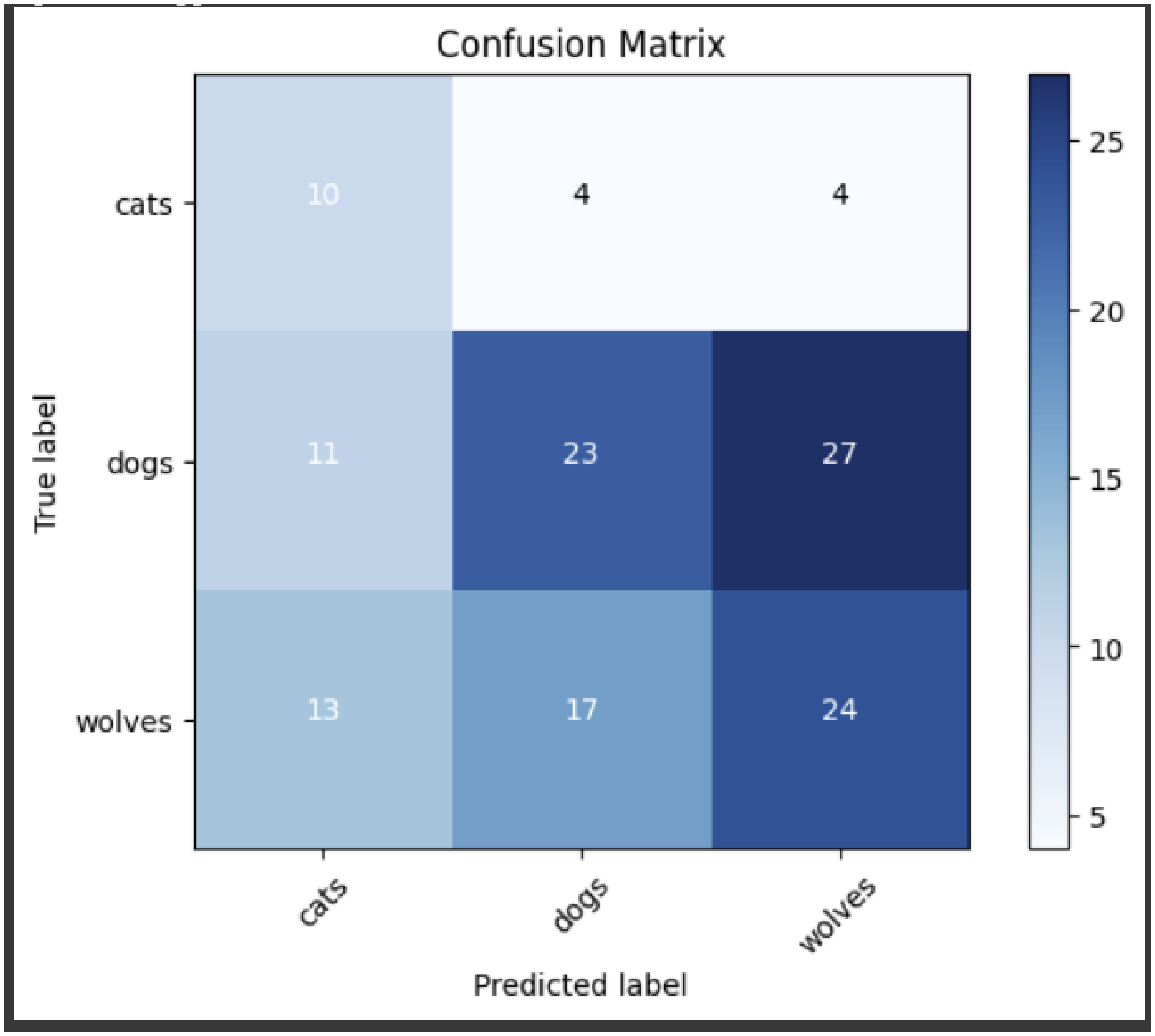
The above screenshot plots the number of epochs on the x-axis and the loss on the y-axis. Despite a few sharp drops in the training loss at certain epochs, the training loss remained relatively stagnant across the number of epochs. Similarly, the loss on the validation curve also remained relatively flat. This may suggest the model has not been trained enough and would benefit from additional training. Perhaps, increasing the number of epochs would help reduce the loss. Furthermore, experimenting with more complex CNN architectures such as two or three convolution/max pooling pairs may help reduce the loss.

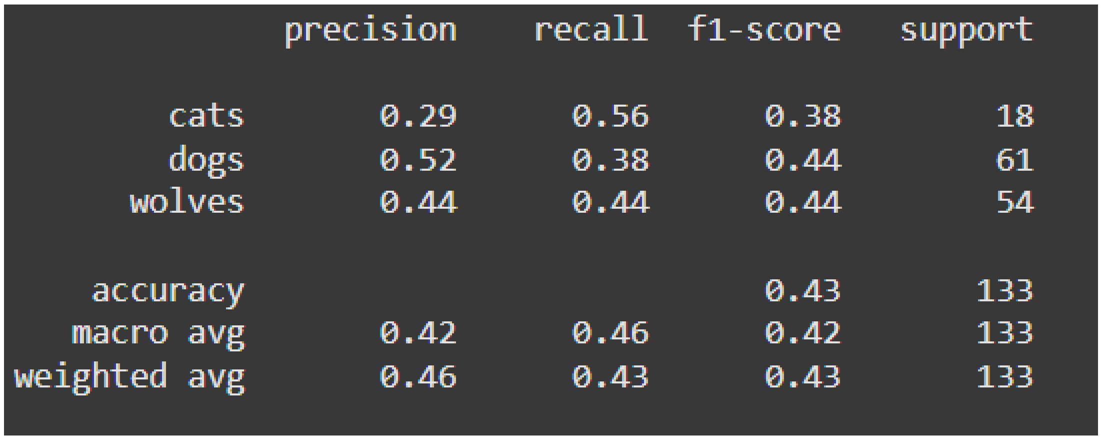


```
print(f"Total training time: {end_time - start_time:.2f} seconds using a T4 GPU")
```

```
Total training time: 109.40 seconds using a T4 GPU
```

As shown above, the total training time using 1 convolution/max pooling layer in the CNN was approximately 110 seconds. The training was done on a T4 GPU in Google Colab.





```
              precision    recall  f1-score   support

        cats       0.29      0.56      0.38        18
        dogs       0.52      0.38      0.44        61
      wolves       0.44      0.44      0.44        54

    accuracy                           0.43       133
   macro avg       0.42      0.46      0.42       133
weighted avg       0.46      0.43      0.43       133
```

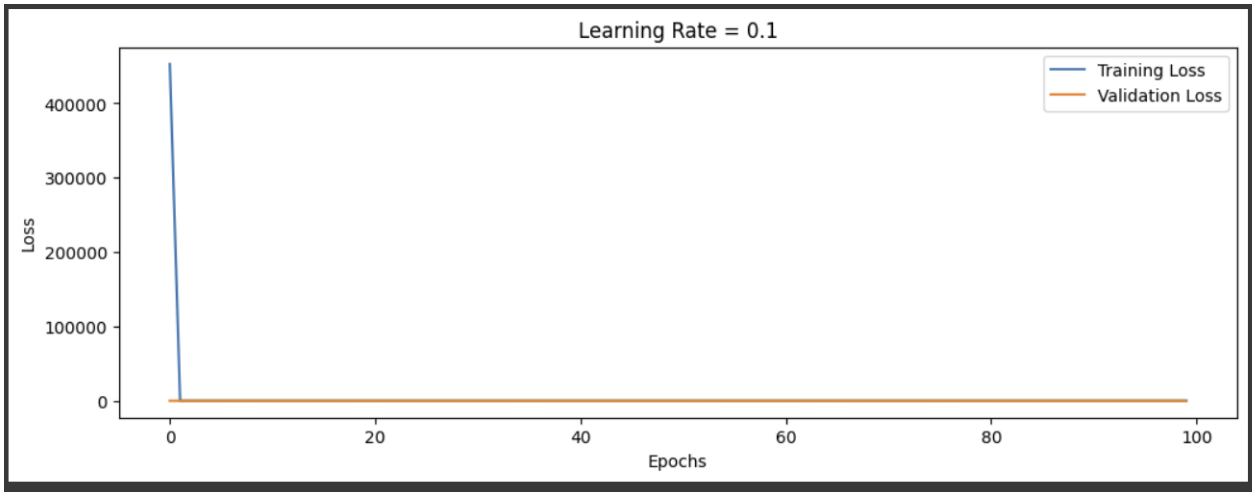
As shown above, the precision for the cat, dog, and wolf classes are 0.29, 0.52, and 0.44 respectively. Precision measures the proportion of correctly predicted positives divided by total predicted positives. In this case, we can observe the dog class had the highest precision. This indicates the model was most precise in correctly classifying dog images among its predictions.

The recall for the cat, dog, and wolf classes are 0.56, 0.38, and 0.44 respectively. Recall measures the proportion of correctly predicted positives divided by the total actual positives. In this case, we can observe that the dog class had the lowest recall among all the classes. This indicates there may have been many false negatives for the dog class, many images that were dogs were incorrectly classified as not dogs. Conversely, the cat and wolf classes achieved higher recall scores, indicating fewer false negatives for these categories.

The F1 score for the cat, dog, and wolf classes are respectively 0.38, 0.44, and 0.44. F1 is another evaluation metric which measures the harmonic mean between precision and recall. In this case, we can observe that the dog and wolf classes had the highest F1 score, which indicate they do a reasonable job at both precision and recall. On the other hand, the cat class has the lowest F1 score, suggesting the model struggles with balancing precision and recall for this class. Further steps like increasing the number of cat samples or applying data augmentation specifically for cats may help improve the F1 score.

Experimentation with various Hyperparameters[Batch Size, Epochs, and Learning Rates]

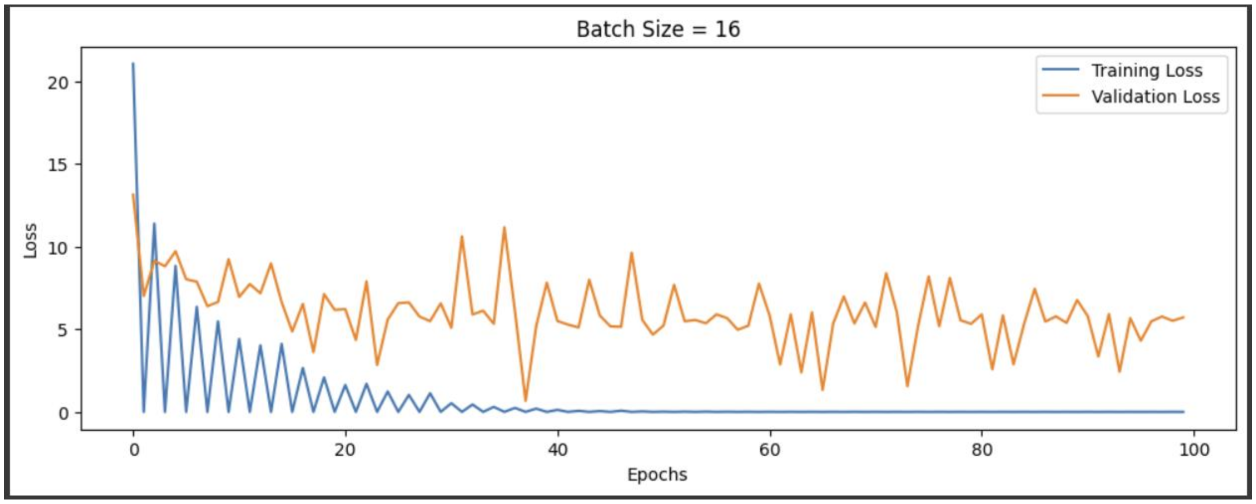
Learning Rate = 0.1



As shown above, a learning rate of 0.1 was used during training. There is a sharp decline in the training loss in the first few epochs, which may indicate aggressive updates to the weights because of too large of a step size. The high learning rate may cause the model to overshoot the optimal weights, which prevents proper convergence. In regard to the validation curve, it remains stagnant as the number of epochs increase. Adjusting the learning rate to be smaller or tuning other hyperparameters may help smoothen out the curves.

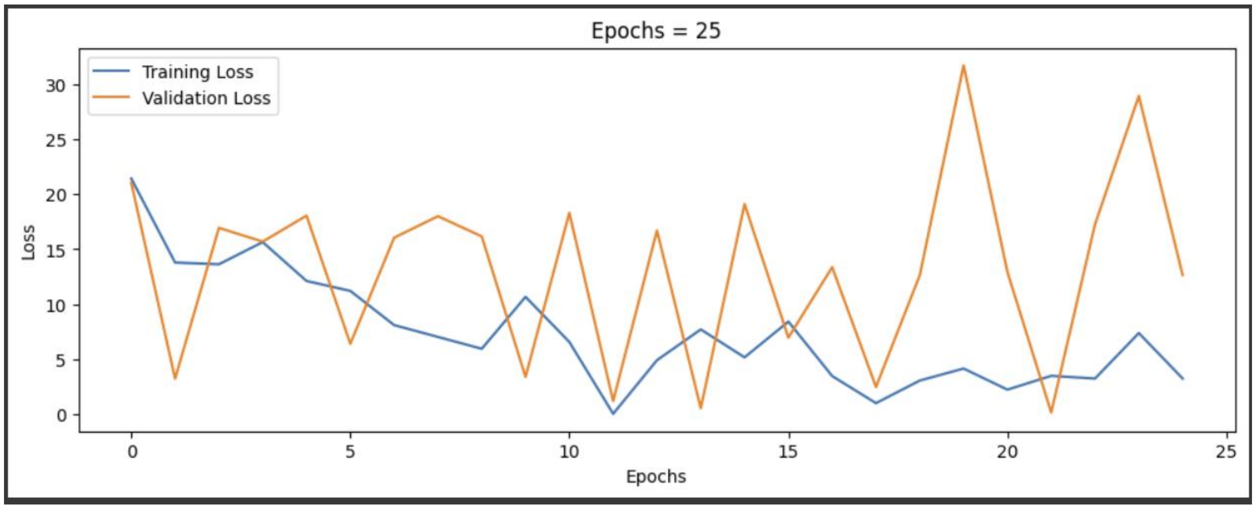
Optimal Learning Rate(0.000001) was used in the remaining portion of this section

Batch Size = 16



As shown above, a batch size of 16 was used during training. Despite fluctuations in the training curve, it decreases as the number of epochs increase. With that said, it plateaus after about epoch 40, which may suggest additional training does not further bring down the loss. In regard to the validation curve, it oscillates frequently, which suggests the model is unable to generalize well to unseen data. Furthermore, smaller batch sizes tend to cause nosier updates to the model weights, which may explain the many fluctuations in the training and validation curves. Modifying the batch size or other hyperparameters may help improve the model's generalization.

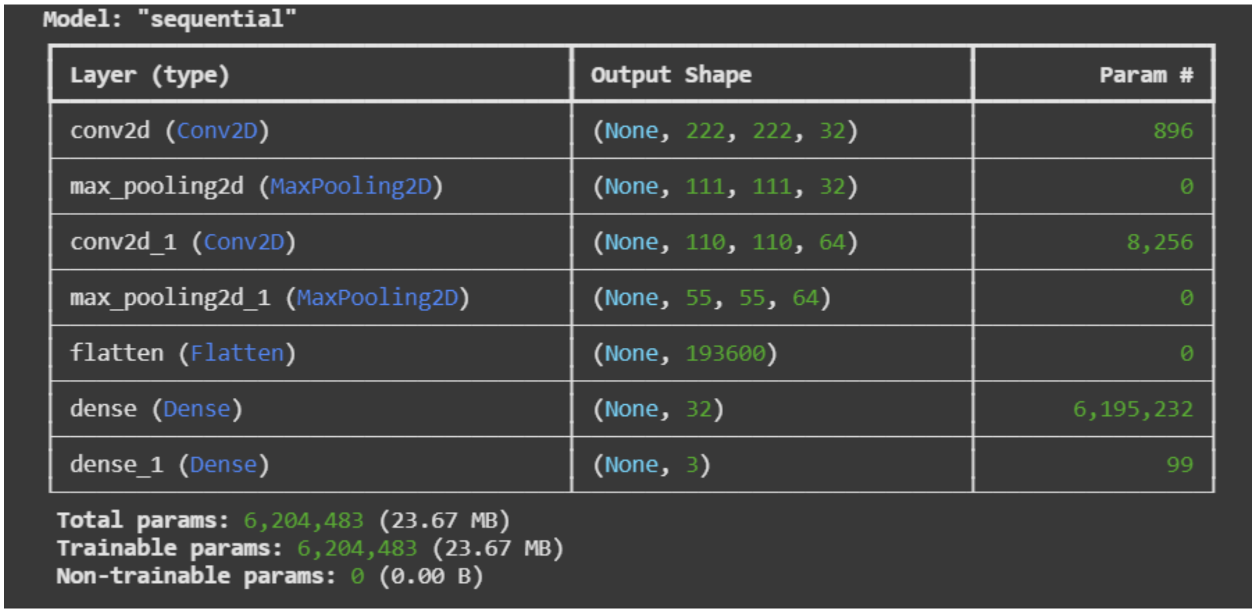
Epochs = 25



As shown above, training was done for 25 epochs. Although the training curve is jagged at many points, it generally decreases as the number of epochs increase. In regard to the validation curve, it is quite sharp at many points and fluctuates. This may be indicative that

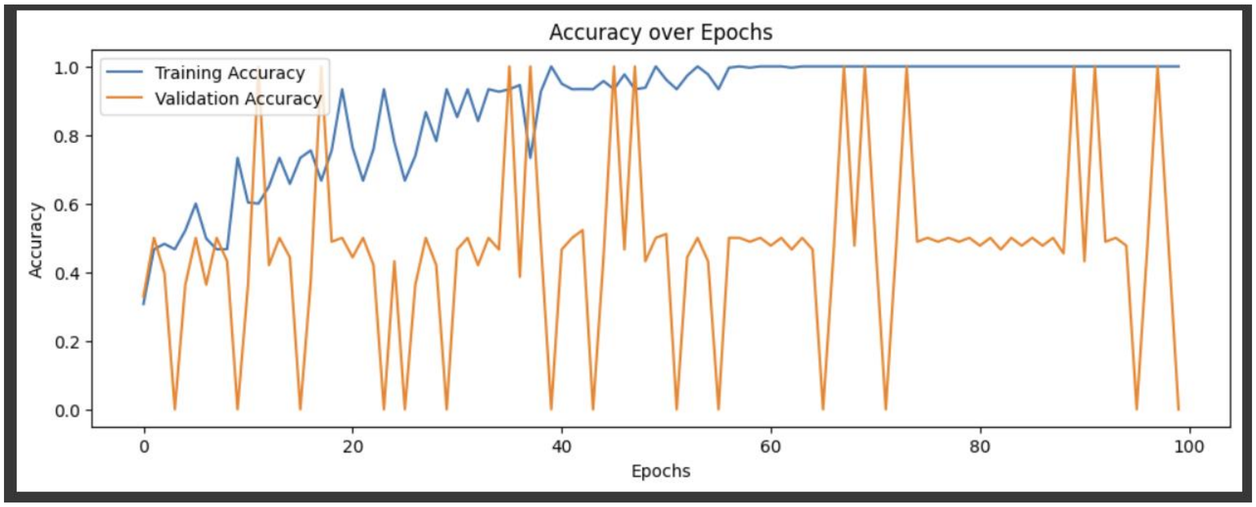
25 epochs is too few for the model to properly learn the patterns in the data. By training for a greater number of epochs and adjusting other hyperparameters, this may help smoothen out both the training and validation curves.

2 Convolution/Max Pooling Pairs

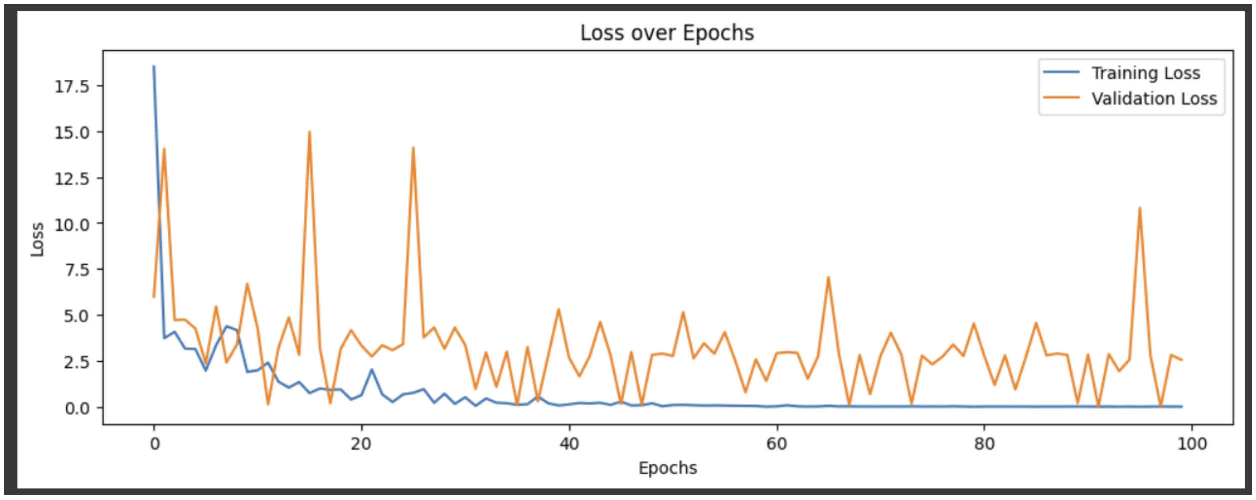


```
Model: "sequential"
┌──────────────────────────────────┬──────────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape             │      Param # │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ conv2d (Conv2D)                  │ (None, 222, 222, 32)     │          896 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 111, 111, 32)     │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ conv2d_1 (Conv2D)                │ (None, 110, 110, 64)     │        8,256 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 55, 55, 64)       │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ flatten (Flatten)                │ (None, 193600)           │            0 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dense (Dense)                    │ (None, 32)               │    6,195,232 │
├──────────────────────────────────┼──────────────────────────┼──────────────┤
│ dense_1 (Dense)                  │ (None, 3)                │           99 │
└──────────────────────────────────┴──────────────────────────┴──────────────┘
 Total params: 6,204,483 (23.67 MB)
 Trainable params: 6,204,483 (23.67 MB)
 Non-trainable params: 0 (0.00 B)
```

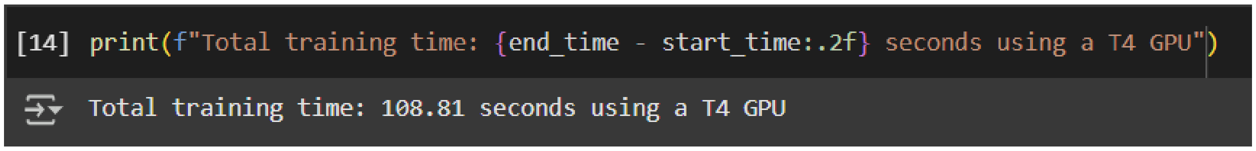
The above model summary indicates that 2 convolution/max pooling pairs are included in the CNN architecture.



As shown above, the x-axis plots the number of epochs, and the y-axis plots the accuracy. The training accuracy tends to generally increase as the number of epochs increase, indicating the model is benefiting from additional training. On the other hand, the validation accuracy is jagged at many points and is oscillating quite frequently. This may be a sign of overfitting because the training accuracy increases while the validation accuracy fluctuates a lot. Regularization techniques such as dropout or L2 regularization may help reduce the complexity of the model.

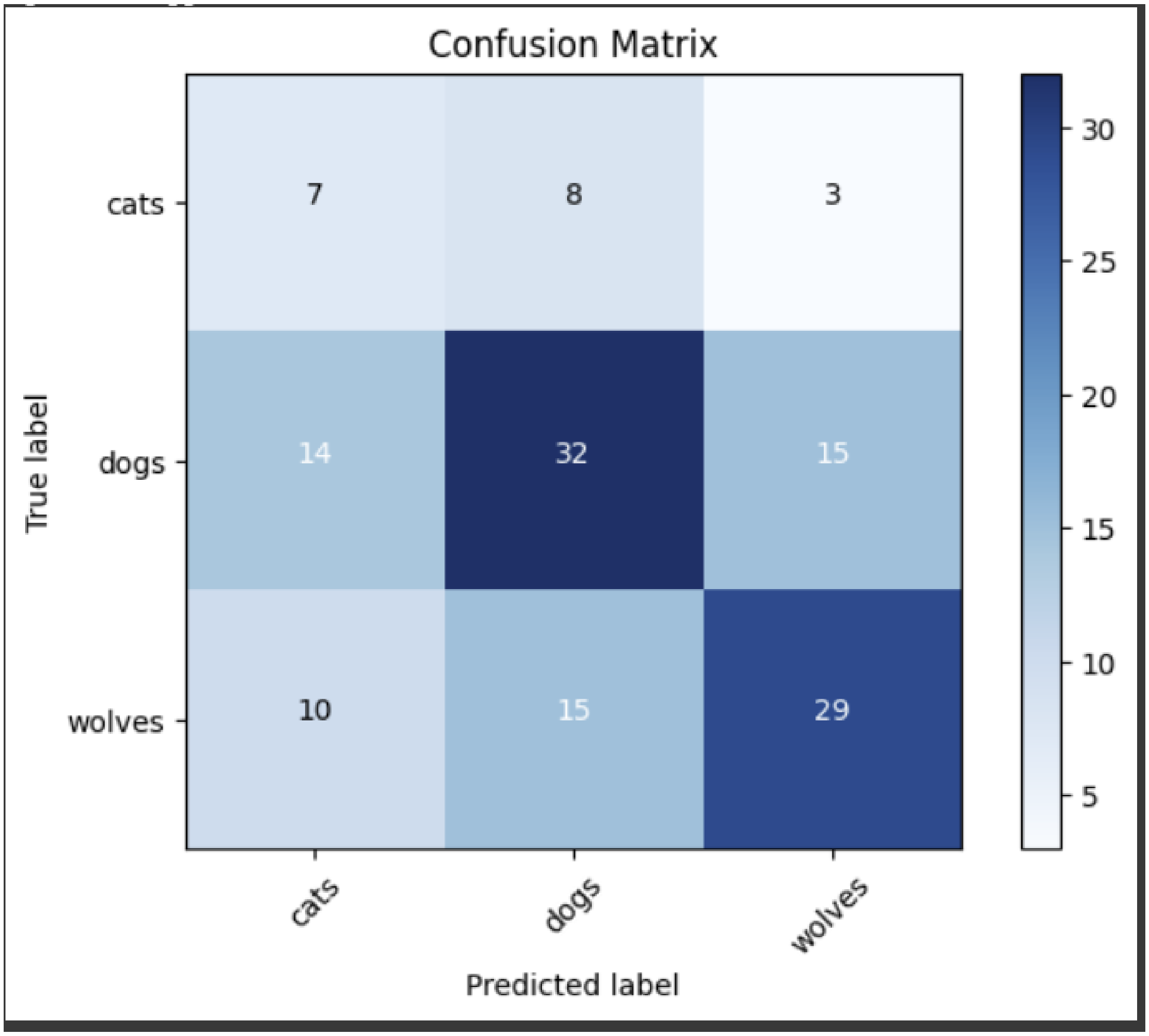


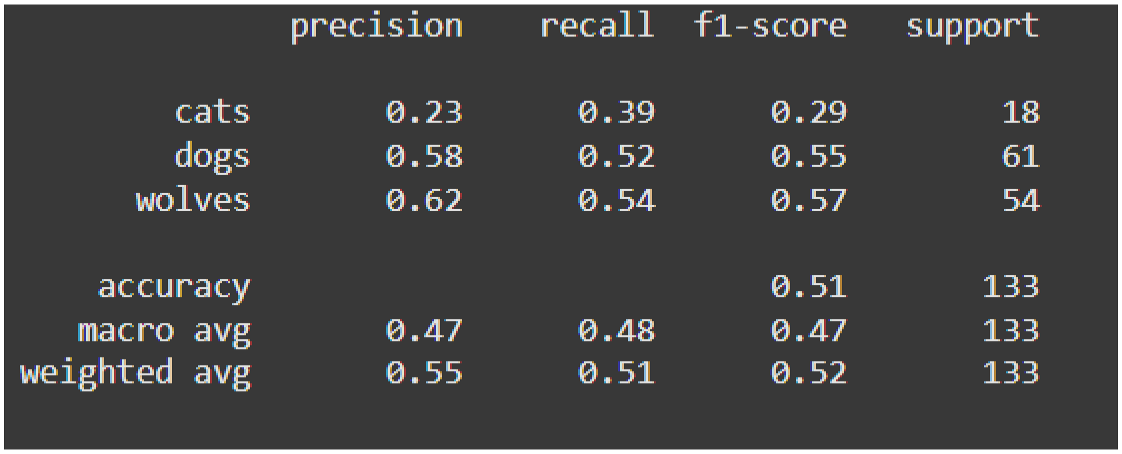
As shown above, the x-axis plots the number of epochs, and the y-axis plots the loss. The training loss generally decreases as the number of epochs increase. After 50 epochs, the training loss remains stagnant which may indicate 100 epochs may be too many to train on. Regarding the validation loss, it is quite jagged and fluctuates at many points. This may suggest the model struggles to generalize to unseen data. To help combat this, perhaps reducing the number of epochs to 50 or adjusting other hyperparameters such as reducing the learning rate may help lower the complexity of the model.



```
[14] print(f"Total training time: {end_time - start_time:.2f} seconds using a T4 GPU")

     Total training time: 108.81 seconds using a T4 GPU
```

As shown above, the total training time using 2 convolution/max pooling layer pairs in the CNN was approximately 110 seconds. The training was done on a T4 GPU in Google Colab.





```
              precision    recall  f1-score   support

        cats       0.23      0.39      0.29        18
        dogs       0.58      0.52      0.55        61
      wolves       0.62      0.54      0.57        54

    accuracy                           0.51       133
   macro avg       0.47      0.48      0.47       133
weighted avg       0.55      0.51      0.52       133
```

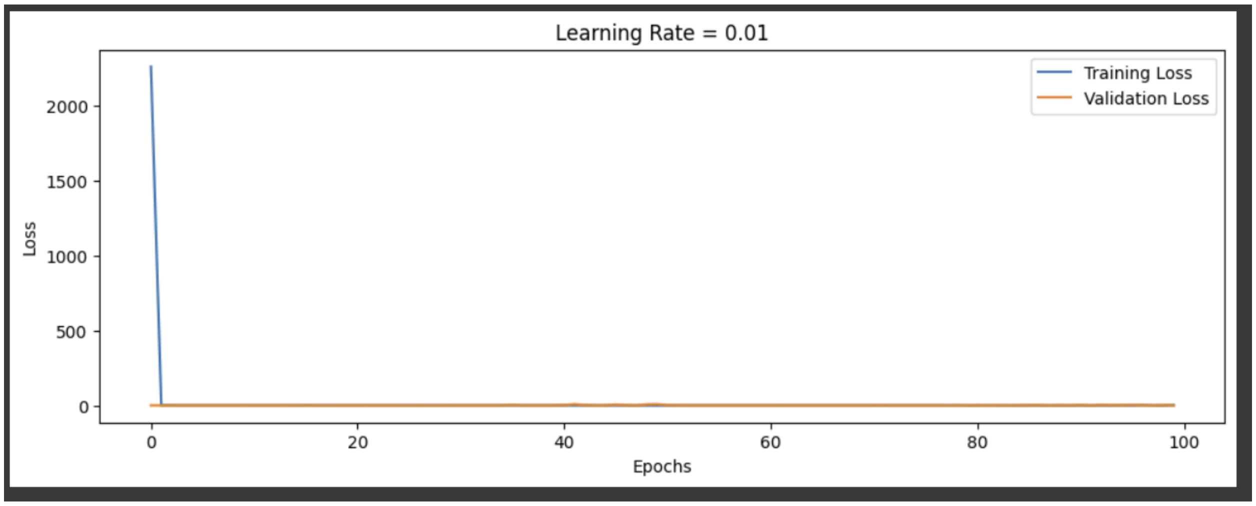
As shown above, the precision for the cat, dog, and wolf classes are 0.23, 0.58, and 0.62 respectively. Precision measures the proportion of correctly predicted positives divided by total predicted positives. In this case, we can observe the wolf class had the highest precision. This indicates the model was most precise in correctly classifying wolf images among its predictions.

The recall for the cat, dog, and wolf classes are 0.39, 0.52, and 0.54 respectively. Recall measures the proportion of correctly predicted positives divided by the total actual positives. In this case, we can observe that the cat class had the lowest recall among all the classes. This indicates there may have been many false negatives for the cat class, many images that were cats were incorrectly classified as not cats. Conversely, the dog and wolf classes achieved similar recall scores, indicating fewer false negatives for these categories.

The F1 score for the cat, dog, and wolf classes are respectively 0.29, 0.55, and 0.57. F1 is another evaluation metric which is the harmonic mean between precision and recall. In this case, we can observe that the wolf class had the highest F1 score, which indicates it does a reasonable job at both precision and recall. On the other hand, the cat class has the lowest F1 score, suggesting the model struggles with balancing precision and recall for this class. Further steps like increasing the number of cat samples or applying data augmentation specifically for cats may help improve the F1 score.

Experimentation with various Hyperparameters[Batch Size, Epochs, and Learning Rates]

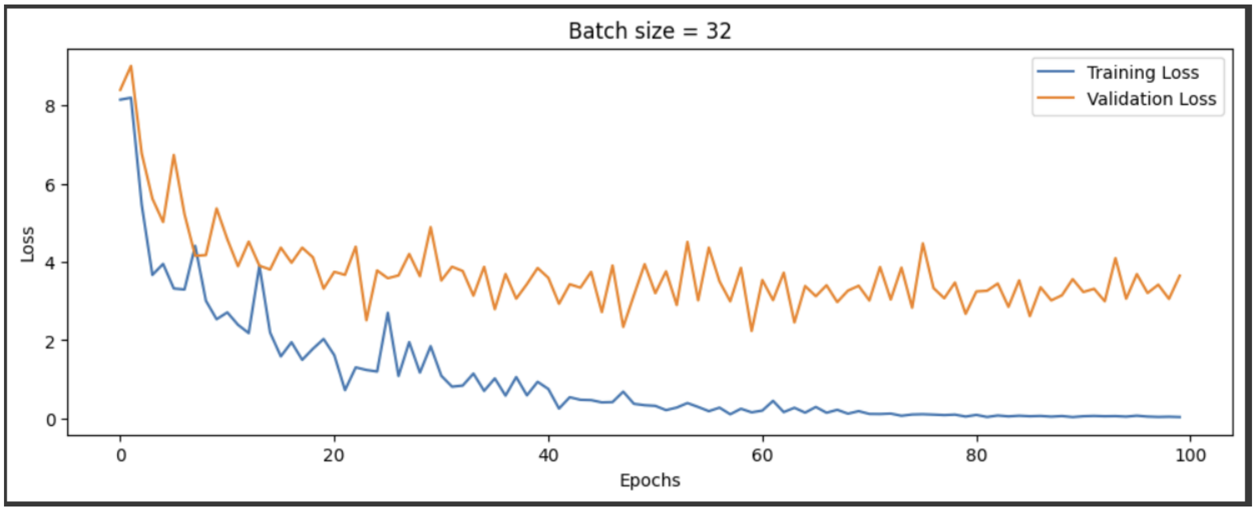
Learning Rate = 0.01



As shown above, a learning rate of 0.01 was used during training. The training loss starts extremely high and suddenly drops to a lower value within the first few epochs. This may be indicative that the model is learning too aggressively due to the high learning rate. In regard to the validation curve, it remains stagnant as the number of epochs increase. This suggests overfitting and the model is unable to generalize well to unseen data. Lowering the learning rate or adjusting other hyperparameters may help the model generalize better.

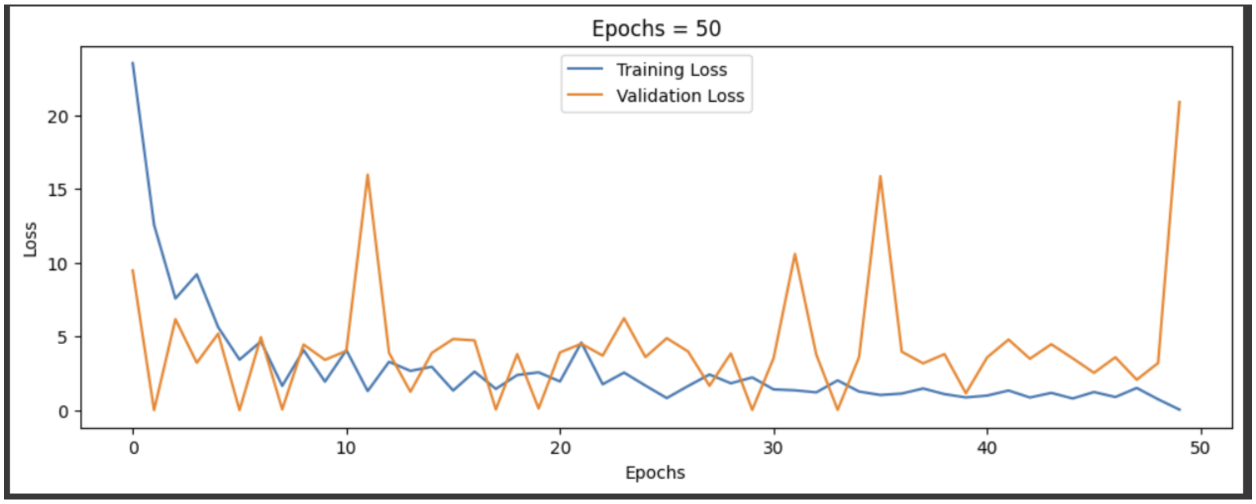
Optimal Learning Rate(0.000001) was used in the remaining portion of this section

Batch Size = 32



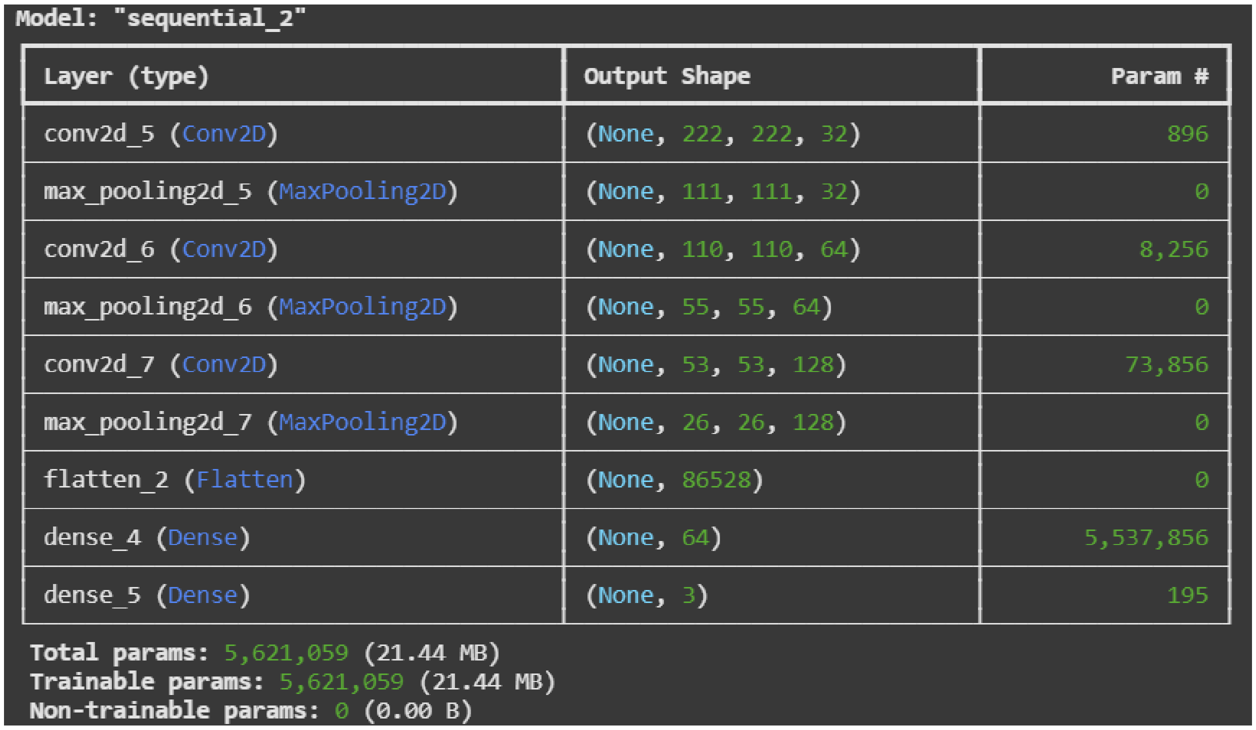
As shown above, a batch size of 32 was used during training. The training loss generally decreases as the number of epochs increase. The validation curve follows a similar trend. Both the training and validation curves are jagged at certain points, which may indicate model instability. Although the learning rate could be reduced further, it may become too small and not converge. Instead, tinkering with other hyperparameters or applying some regularization techniques may help improve model performance.

Epochs = 50



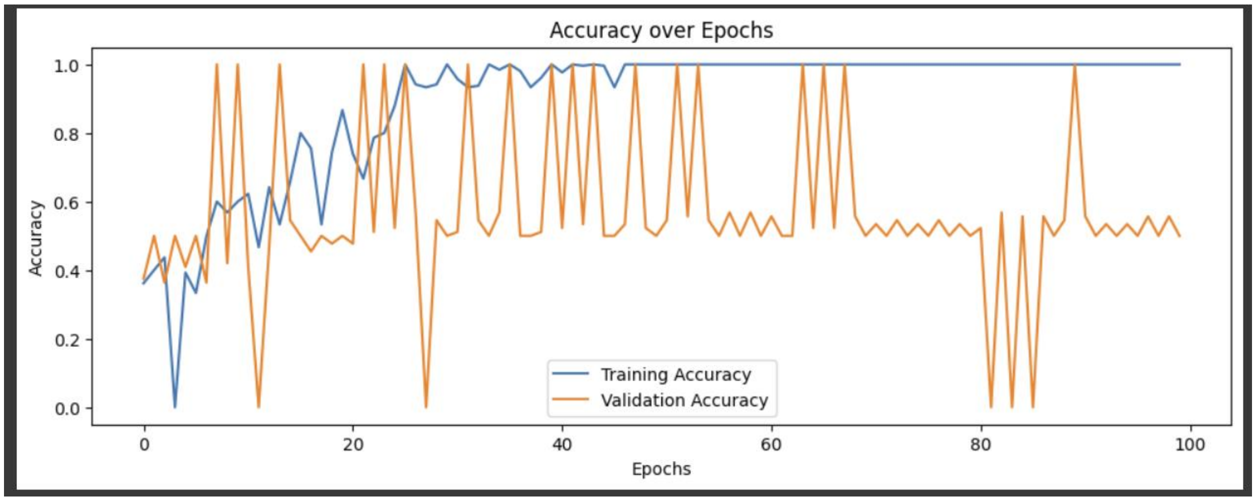
As shown above, training was done for 50 epochs instead of 100 epochs. The training loss did generally decrease as the number of epochs increased. With that said, it was jagged at certain points. The validation curve was significantly oscillating, indicating the model had difficulty generalizing unseen data. This may indicate 50 epochs is too few for the model to fully converge. Increasing the number of epochs while monitoring validation trends could help improve model generalization.

3 Convolution/Max Pooling Pairs

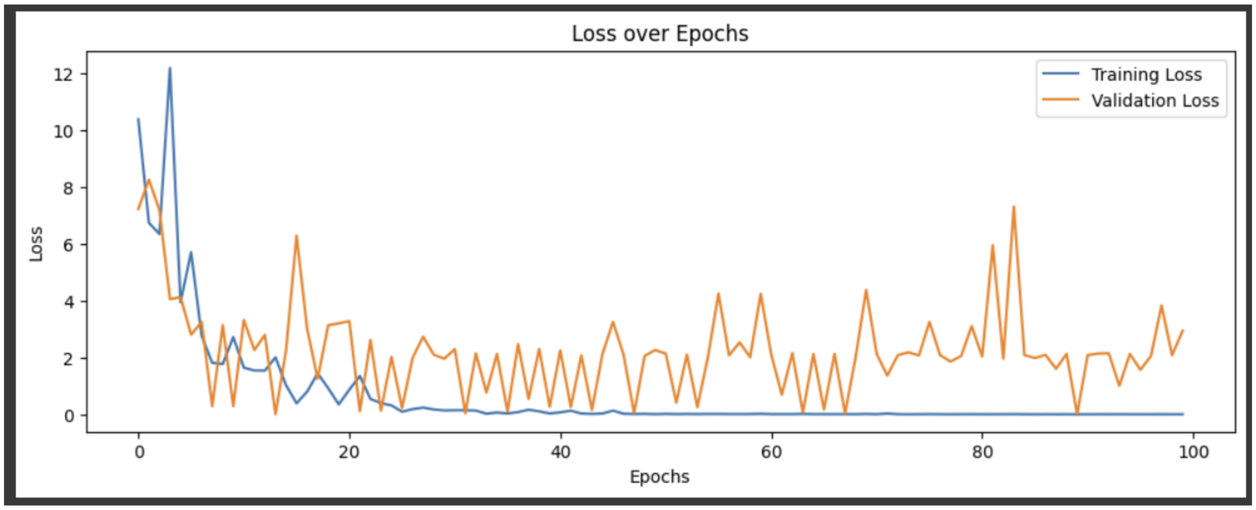


```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_5 (Conv2D)               │ (None, 222, 222, 32)   │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_5 (MaxPooling2D)  │ (None, 111, 111, 32)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_6 (Conv2D)               │ (None, 110, 110, 64)   │         8,256 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_6 (MaxPooling2D)  │ (None, 55, 55, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_7 (Conv2D)               │ (None, 53, 53, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_7 (MaxPooling2D)  │ (None, 26, 26, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_2 (Flatten)             │ (None, 86528)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_4 (Dense)                 │ (None, 64)             │     5,537,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_5 (Dense)                 │ (None, 3)              │           195 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 5,621,059 (21.44 MB)
 Trainable params: 5,621,059 (21.44 MB)
 Non-trainable params: 0 (0.00 B)
```

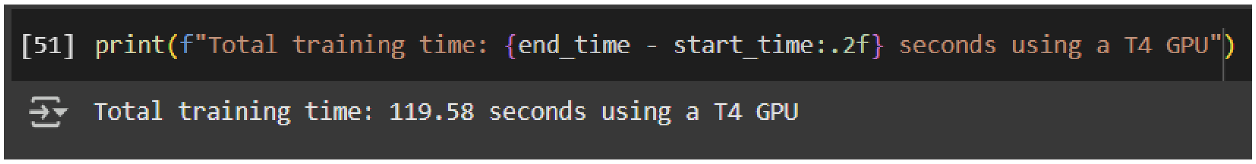
The above model summary indicates that 3 convolution/max pooling pairs are included in the CNN architecture.



As shown above, the x-axis plots the number of epochs, and the y-axis plots the accuracy. The training accuracy increases as the number of epochs increase. The training accuracy remains stagnant after epoch 50. This may suggest that additional epochs do not provide any gain in accuracy. The validation accuracy is highly volatile. This may be a sign the model is starting to overfit and memorize the training data. Consequently, the model may have a difficult time generalizing to unseen data. Reducing the number of epochs or adding some regularization techniques may help combat the overfitting of the model.



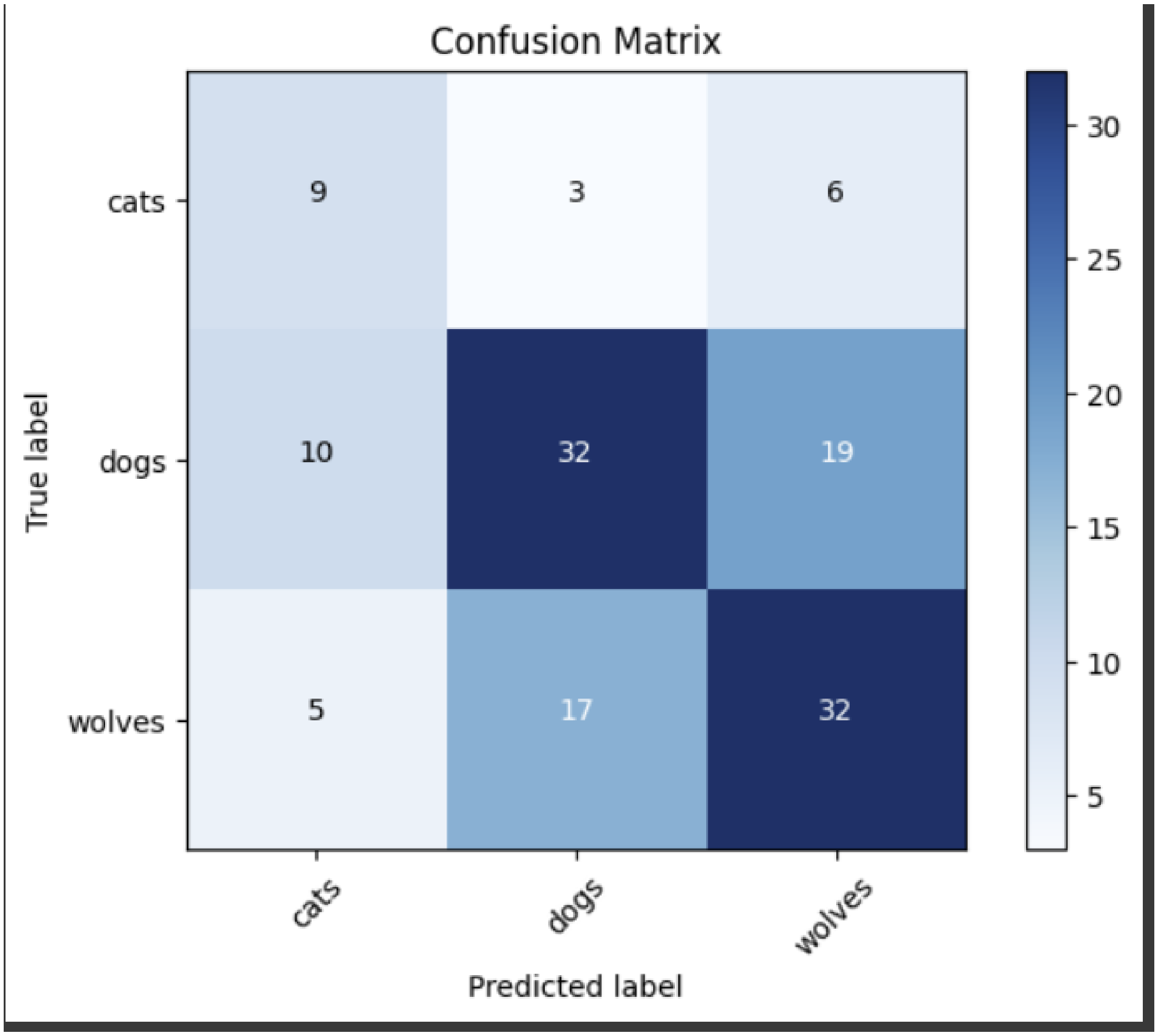
As shown above, the x-axis plots the number of epochs, and the y-axis plots the loss. The training loss generally decreases as the number of epochs increase. The training loss remains stagnant after epoch 50 which may suggest that additional training does not reduce the loss further. Regarding the validation loss, it is quite jagged at certain points and does fluctuate. Dropout or other regularization techniques may be utilized to make the curve for validation loss smoother.

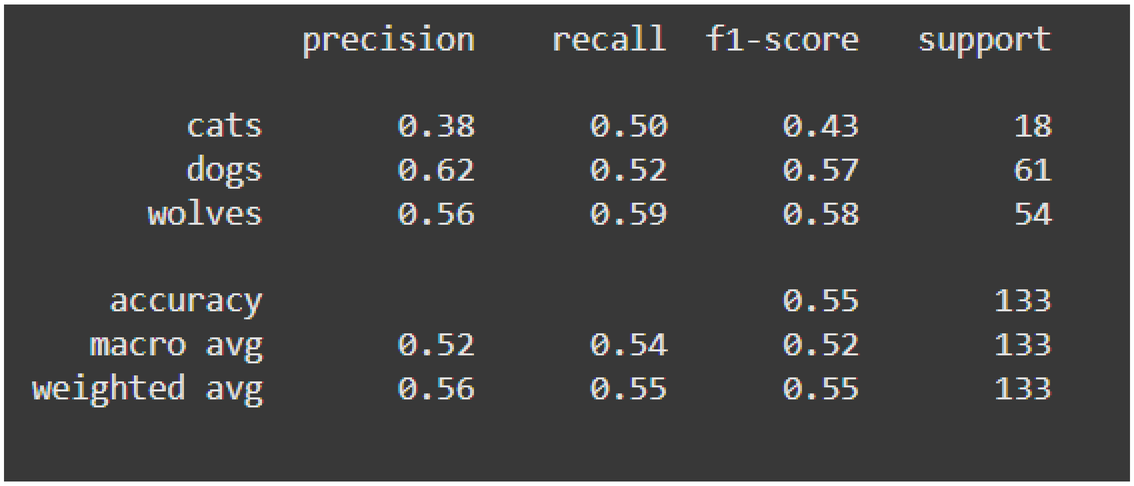


```
[51] print(f"Total training time: {end_time - start_time:.2f} seconds using a T4 GPU")
```

```
Total training time: 119.58 seconds using a T4 GPU
```

As shown above, the total training time using 3 convolution/max pooling layer pairs in the CNN was approximately 120 seconds. The training was done on a T4 GPU in Google Colab. Since this CNN model was more complex than the previous two, it makes sense that it took more time to train.





```
              precision    recall  f1-score   support

        cats       0.38      0.50      0.43        18
        dogs       0.62      0.52      0.57        61
      wolves       0.56      0.59      0.58        54

    accuracy                           0.55       133
   macro avg       0.52      0.54      0.52       133
weighted avg       0.56      0.55      0.55       133
```

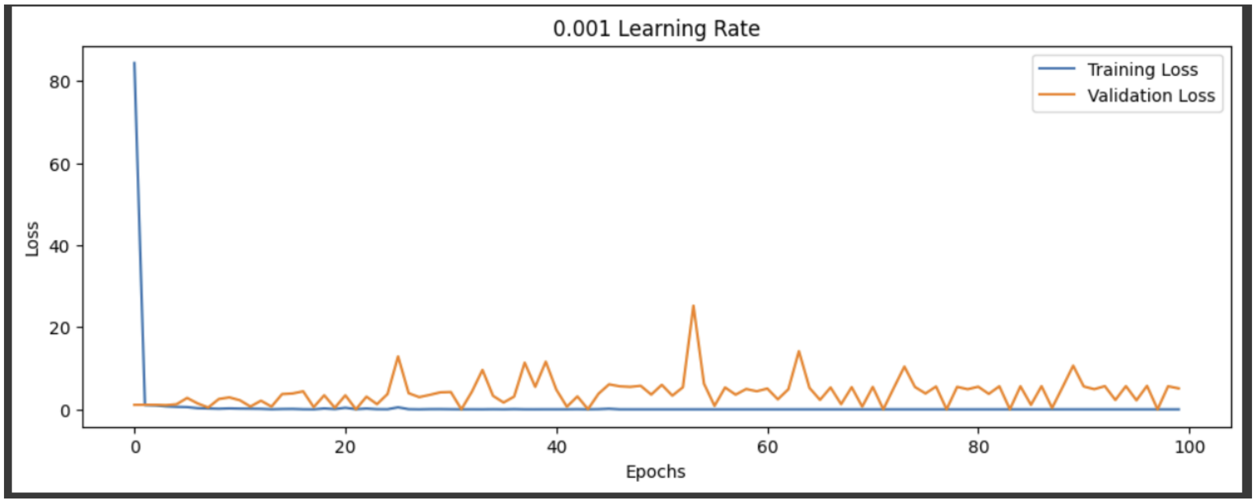
As shown above, the precision for the cat, dog, and wolf classes are 0.38, 0.62, and 0.56 respectively. Precision measures the proportion of correctly predicted positives divided by total predicted positives. In this case, we can observe the dog and wolf classes had higher precision compared to the cat class.

The recall for the cat, dog, and wolf classes are 0.5, 0.52, and 0.59 respectively. Recall measures the proportion of correctly predicted positives divided by the total actual positives. In this case, we can observe that all three classes had relatively the same recall, indicating that the model is moderately effective in identifying true positives across all classes. With that said, the wolf class had a slightly higher recall relative to the other classes.

The F1 score for the cat, dog, and wolf classes are respectively 0.43, 0.57, and 0.58. F1 is another evaluation metric which is the harmonic mean between precision and recall. In this case, we can observe that the wolf class had the highest F1 score, which indicates it does a reasonable job at both precision and recall. On the other hand, the cat class has the lowest F1 score, suggesting the model struggles with balancing precision and recall for this class. Further steps like increasing the number of cat samples or applying data augmentation specifically for cats may help improve the F1 score.

Experimentation with various Hyperparameters[Batch Size, Epochs, and Learning Rates]

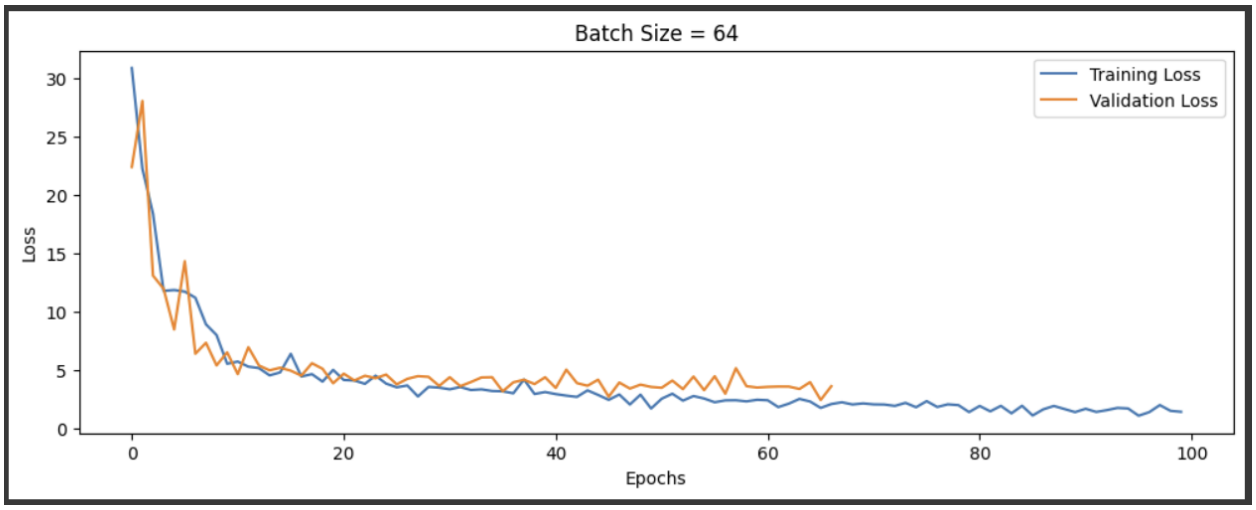
Learning Rate = 0.001



As shown above, a learning rate of 0.001 was used. As the number of epochs increase, the loss remains stagnant. In regard to the validation curve, the curve is significantly fluctuating. In addition, it is quite jagged at certain points in the graph. This may be indicative that the learning rate is too large, and the model is overshooting and is unable to converge. Perhaps making the learning rate smaller may smoothen out the curves.

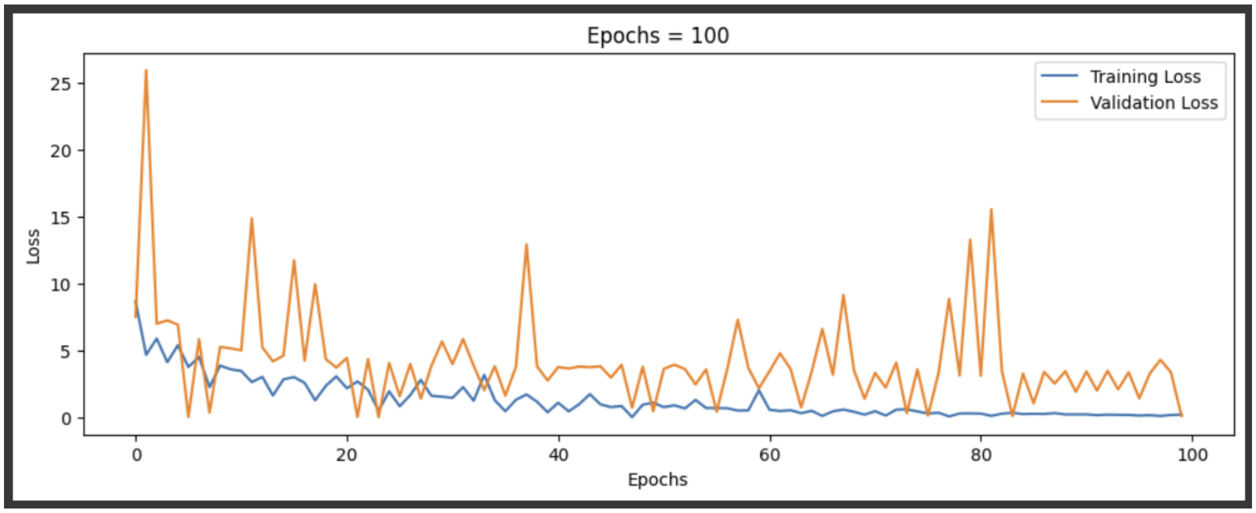
Optimal Learning Rate(0.000001) was used in the remaining portion of this section

Batch Size = 64



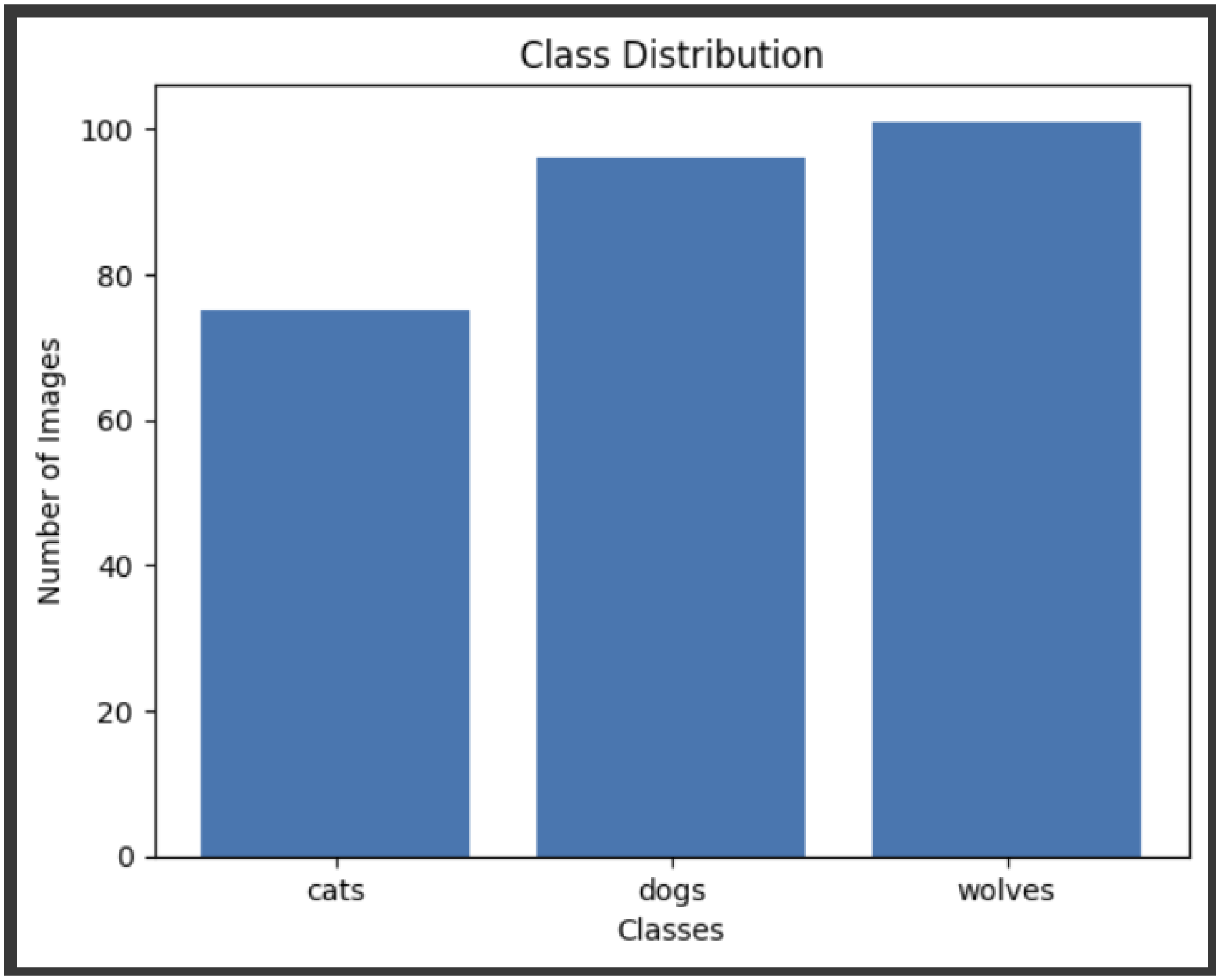
As shown above, a batch size of 64 was used during training with a 3 convolution/max pooling pair CNN. The training loss steadily decreased as the number of epochs increased, showing smooth convergence. The validation loss followed a similar trend with minor fluctuations, which indicates reasonable generalization to unseen data. This suggests that a batch size of 64 is effective for stabilizing the model training process while maintaining good validation performance. However, slight jaggedness in the validation loss could indicate room for further optimization, such as experimenting with regularization or smaller batch sizes.

Epochs = 100



As shown above, training was performed for 100 epochs. The training loss does generally decrease as the number of epochs increase. The training loss plateaus after 80 epochs, which may suggest that additional epochs may make the model overfit. In regard to the validation curve, it is oscillating and quite jagged at many points. This is a sign of overfitting. To combat this, training for a smaller number of epochs or applying regularization techniques such as dropout or L2 regularization may help improve model generalization.

Balanced Dataset Analysis



As shown above, the x-axis plots the classes included in the dataset and the y-axis plots the number of images of each class. The cat class includes 75 images. The dog class includes 96 images. The wolf class includes 101 images. Overall, there is roughly an equal distribution among all the classes. With that said, the dog and wolf classes have slightly more images than the cat class. If there is significant imbalance in the dataset, evaluation metrics such as accuracy can be misleading because the model may achieve a high accuracy just by predicting the majority class. To combat this, techniques such as data augmentation can be used to produce more images of the minority class to have a more evenly distributed dataset.

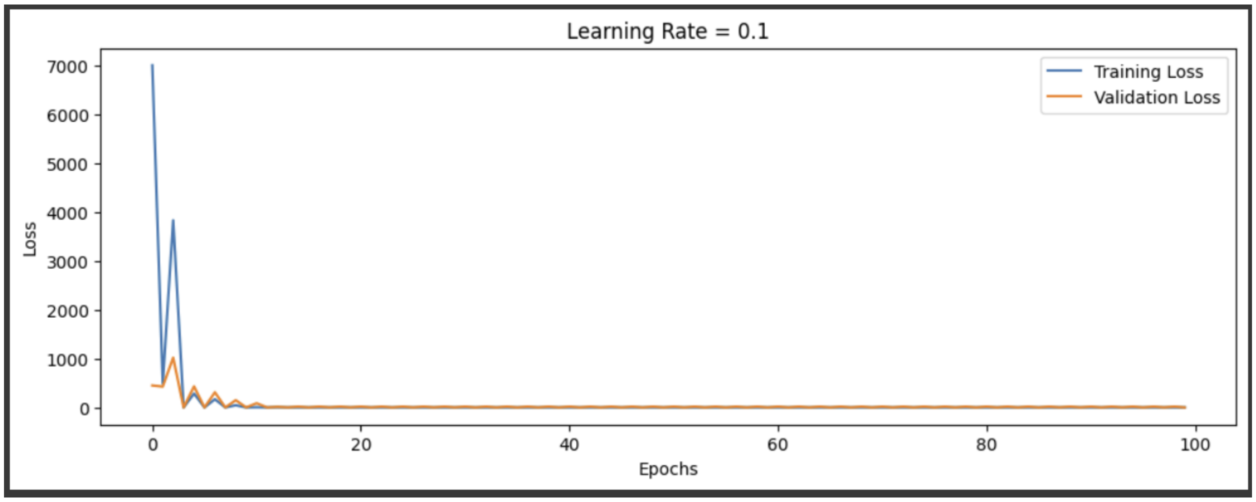
Task 3

For this task, I chose VGG16 because it is a widely used pre-trained CNN model, known for its strong performance on image classification tasks. Its architecture is simple yet effective, and it has been pre-trained on ImageNet, making it a good candidate for transfer learning. In addition, VGG16 makes use of stacked convolutions, the use of multiple convolution layers before a max pooling layer, making its architecture different from traditional CNN models.

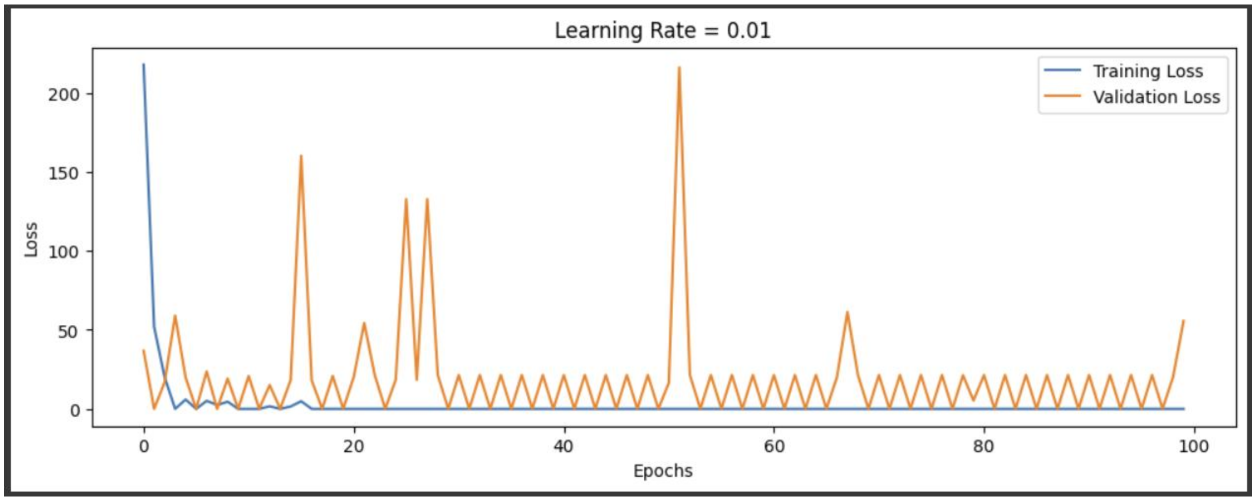
I performed feature extraction by freezing the pretrained layers and training only the custom top layers.

Experimentation with various Hyperparameters[Batch Size, Epochs, and Learning Rates]

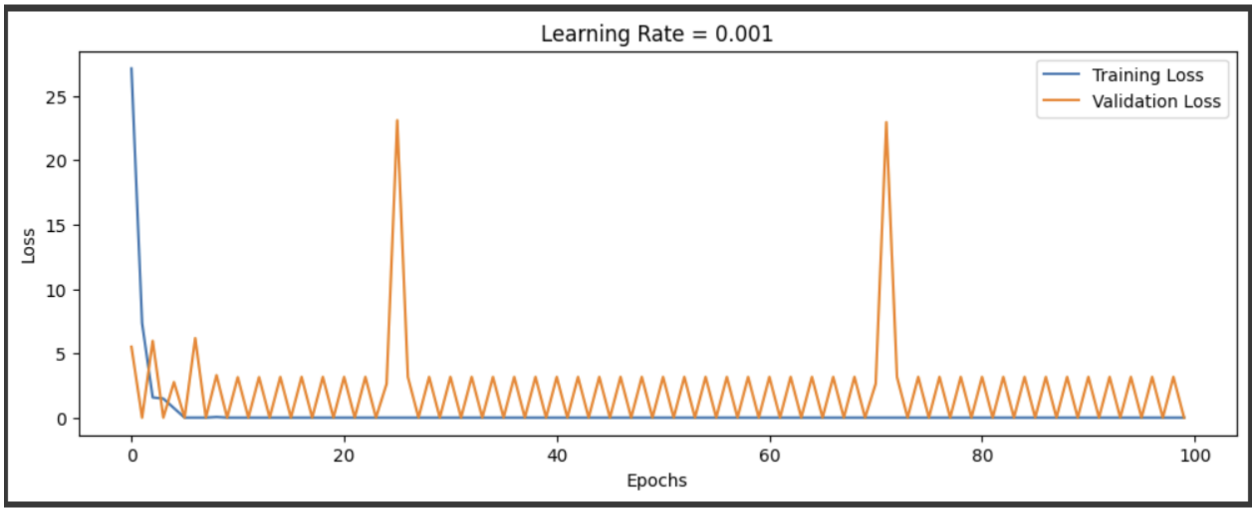
Learning Rate = 0.1



As shown above, using the VGG16 model, there are sharp drops in both the training and validation curves due to the large learning rate of 0.1 used during training. Despite this, the accuracy on the test set is still 0.83. This reflects that although the hyperparameters may be suboptimal, VGG16's pretrained layers contribute significantly to feature extraction, making the model robust, nonetheless.

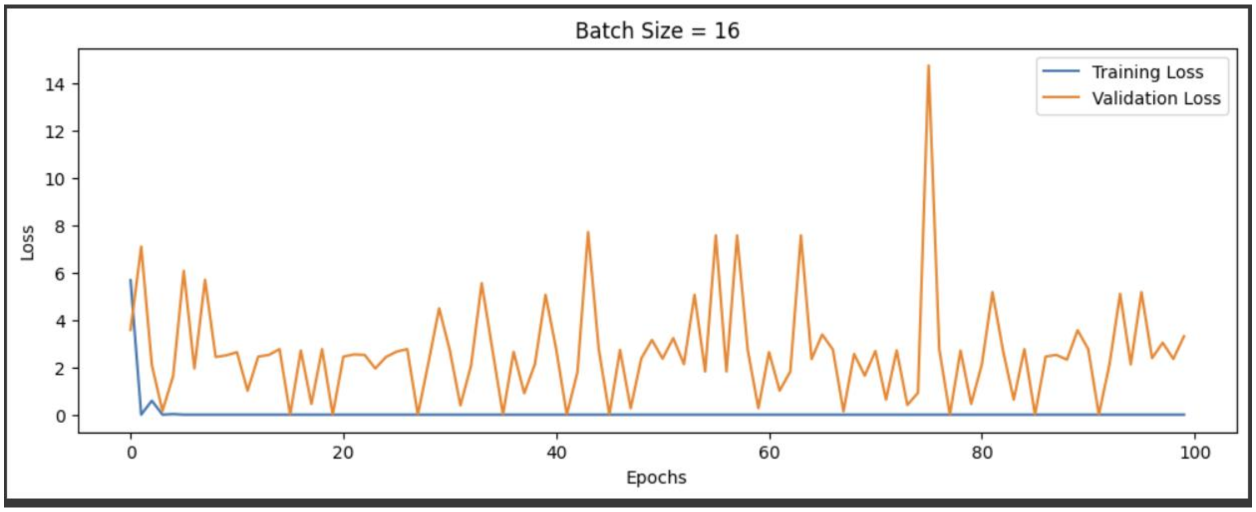


As shown above, a learning rate of 0.01 was used during training. The training curve exhibits a sharp reduction early on, while the validation curve oscillates significantly throughout. This suggests that the learning rate may still be too large for optimal stability. Applying a smaller learning rate might yield smoother and more consistent curves. Despite these observations, the model still achieves 0.89 accuracy on the test set, showcasing the strength of the VGG16 pretrained layers in feature extraction.



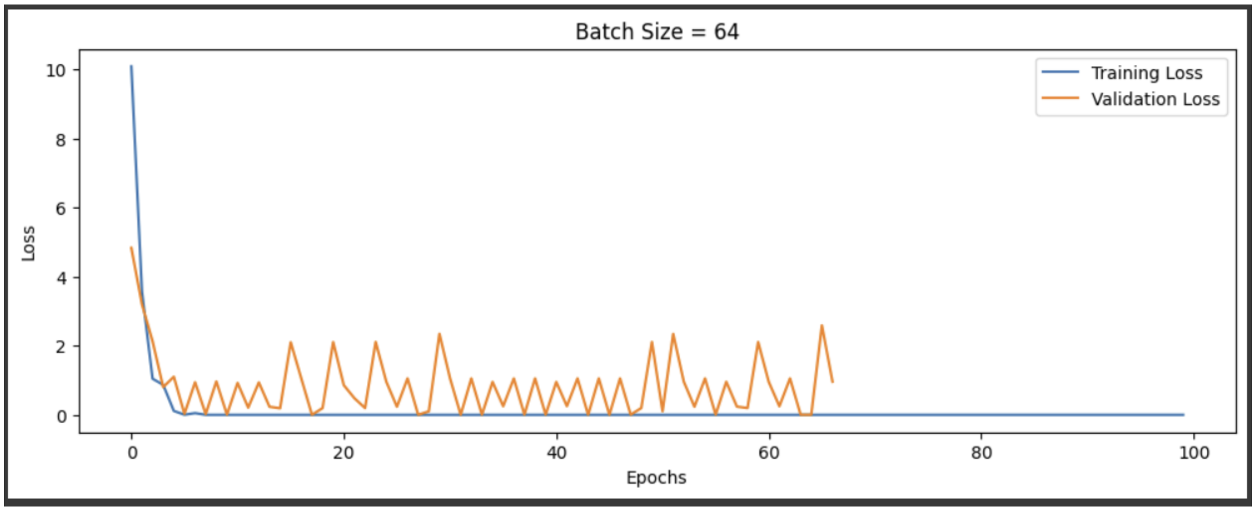
As shown above, a learning rate of 0.001 was used during training. The training curve decreases rapidly in the initial epochs and stabilizes to a near-constant value. However, the validation curve exhibits significant oscillations, with occasional sharp spikes, indicating instability in generalization. Despite this, the model still achieves 0.92 accuracy on the test set. Although the learning rate could be made smaller, it may become too small such that the model may not converge. Hence, a learning rate of 0.001 will be used for the remaining portion of this section.

Batch Size = 16



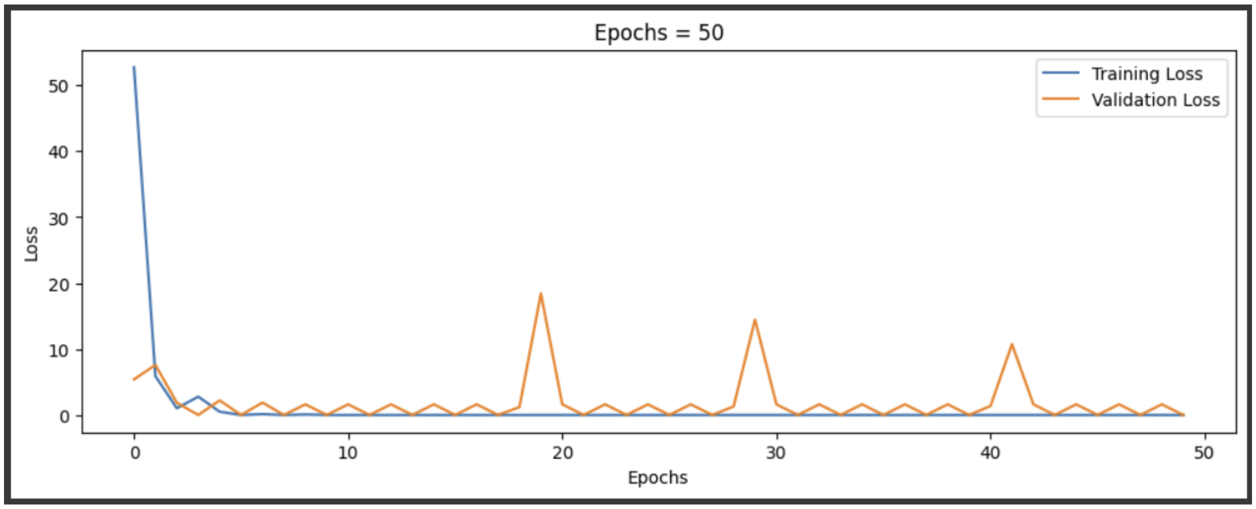
As shown above, training was done with a batch size of 16. The accuracy on the test set was 0.89. The training loss sharply decreased earlier on in the number of epochs and remained flat afterwards. In regard to the validation curve, it is significantly fluctuating, which may indicate that a batch size of 16 is too small for the model to grasp the pattern and generalize. Perhaps, increasing the batch size or modifying other hyperparameters may help with model generalization.

Batch Size = 64



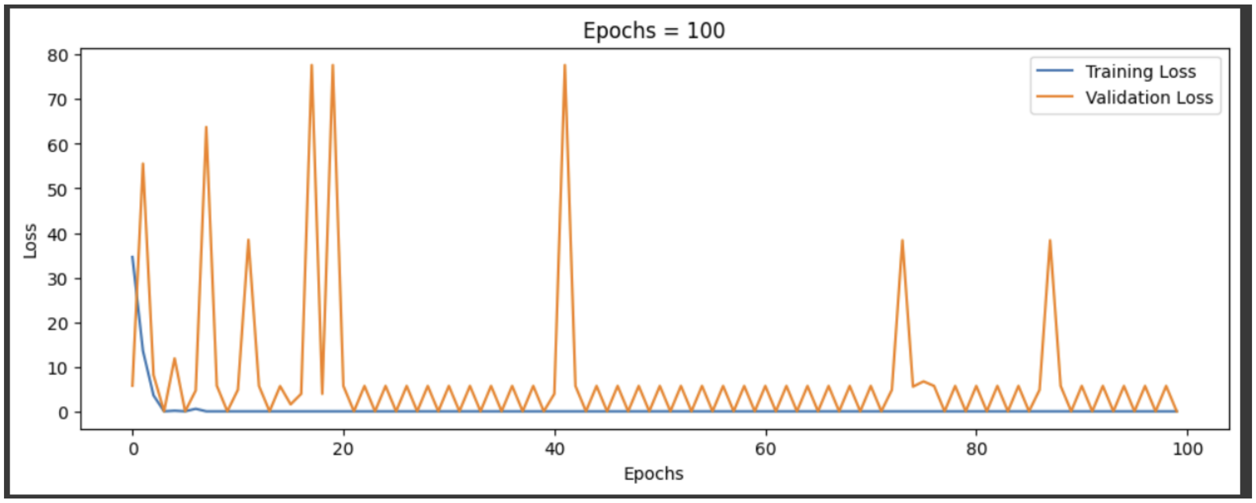
As shown above, training was performed with a batch size of 64, achieving an accuracy of 0.88 on the test set. The training loss sharply decreased during the initial epochs and remained stable afterward. Similarly, the validation curve also exhibited a sharp decrease early on and displayed less fluctuation compared to the case with a batch size of 16. This observation suggests that the larger batch size helps the model better capture patterns and generalize effectively to unseen data.

Epochs = 50



As shown above, training was done for 50 epochs, achieving 0.95 accuracy on the test set. The training curve sharply drops early on in the number of epochs. In regard to the validation curve, it does spike a couple of times across the number of epochs, which may indicate some instability in the model. Perhaps, increasing the number of epochs may help smoothen out the curves.

Epochs = 100



As shown above, training was done for 100 epochs, achieving 0.9 accuracy on the test set. The training curve soon plateaued early on in the number of epochs. The validation curve had multiple sharp spikes, which may indicate that 100 epochs may be too much. Perhaps, reducing the number of epochs or adjusting other hyperparameters may help smoothen out the curves.

Final Summary of Findings

Overall, VGG16 performed well on this classification task, achieving reasonable accuracy on the test set regardless of the hyperparameters used.

Regarding the learning rate, values of 0.1, 0.01, and 0.001 were tested to evaluate their effect on performance. Among these, a learning rate of 0.001 yielded the best accuracy of 0.92, highlighting that a moderate learning rate was optimal for fine-tuning the model.

For batch size, both small (16) and large (64) batch sizes were experimented with. The results indicated that a larger batch size of 64 produced stronger results, likely because it allowed the model to better capture patterns in the data.

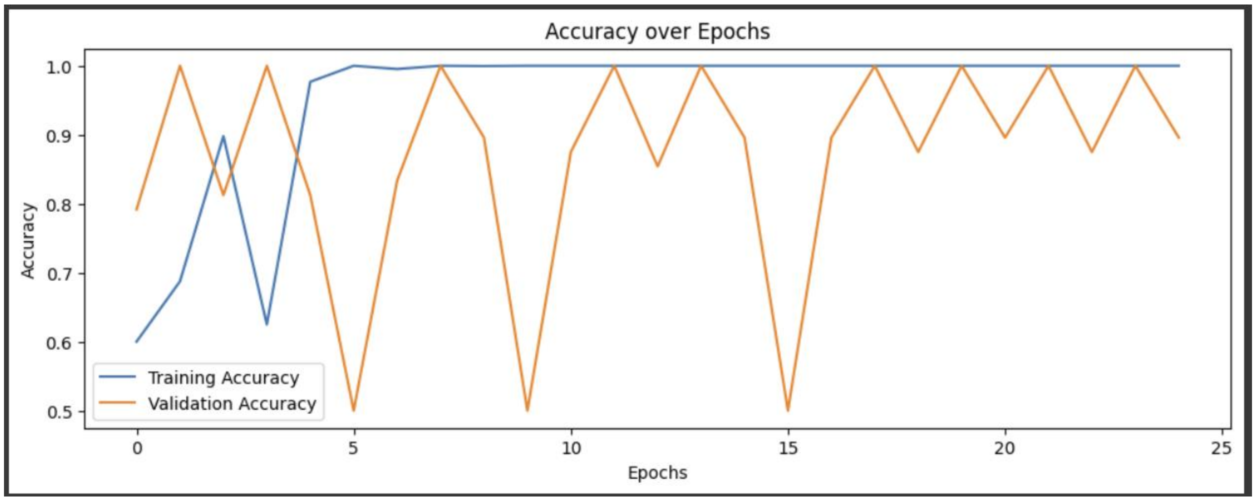
Regarding epochs, training was conducted for 50 and 100 epochs. Both runs exhibited spikes in their validation curves, suggesting instability. This may indicate that the optimal number of epochs falls somewhere between 50 and 100 to prevent overfitting while allowing sufficient learning.

In conclusion, transfer learning with VGG16 proved highly effective for the cat, dog, and wolf classification task. By fine-tuning hyperparameters based on these findings, VGG16 has the potential to achieve even stronger classification performance.

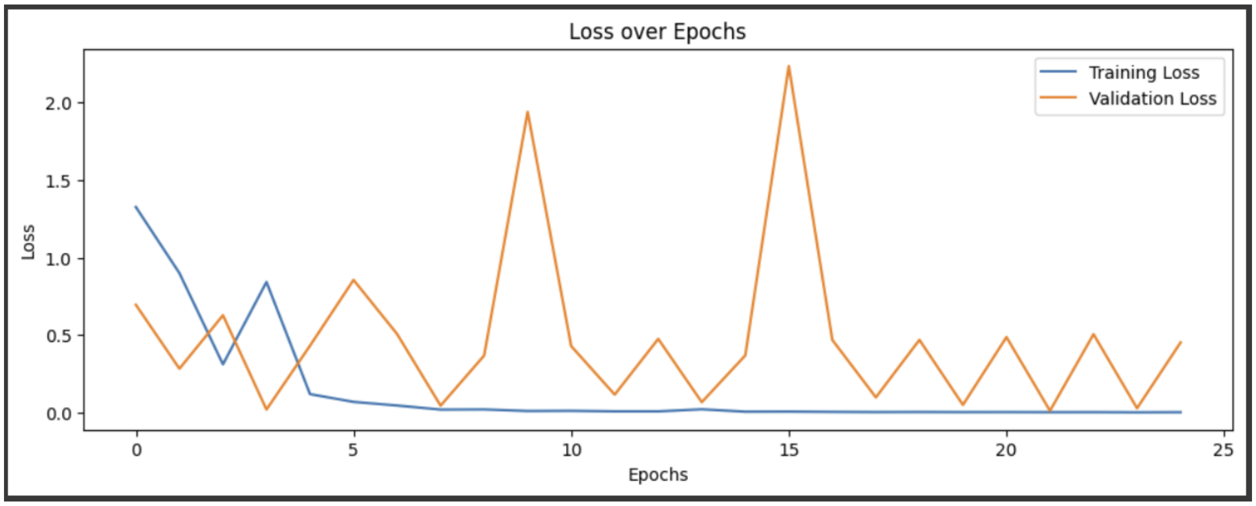
Task 4

For this task, I worked with the 10 Big Cats of the Wild dataset available on Kaggle. The dataset includes images of various animals including cheetahs, pumas, lions, etc. There are 10 classes in this dataset. The dataset consists of 133 images for training, 50 images for validation, and 50 images for testing per class. I chose the 10 Big Cats of the Wild dataset because it is more diverse than Cat, Dog, and Wolf dataset used earlier.

I utilized VGG16, the pretrained CNN, for classification on this dataset. The final dense layer of the architecture was modified to accommodate 10 classes instead of 1000 classes that VGG16 was originally trained on. I trained for 25 epochs with a batch size of 16.



As shown above, the x-axis plots the number of epochs, and the y-axis plots the accuracy. The accuracy increases as the number of epochs increase. The validation curve oscillates, which may indicate some model instability. Perhaps training for a larger number of epochs may help smoothen out the curves.



As shown above, the x-axis plots the number of epochs, and the y-axis plots the loss. The training loss decreases as the number of epochs increase. The validation loss fluctuates as the number of epochs increase, which may indicate model instability. Training for more epochs or applying some regularization techniques may help combat these issues.

Conclusion

In this assignment, transfer learning was successfully applied using VGG16 to classify images of animals, first with a 3-class dataset (cats, dogs, wolves) and then extended to a 10-class dataset (big cats of the wild). Through experimentation with learning rates, batch sizes, and epochs, valuable insights were gained about how hyperparameters influence model performance. In the case of the Big Cats of the Wild dataset, the model achieved 0.875 accuracy on the test set. This demonstrates the effectiveness of transfer learning in handling complex multi-class classification tasks.