cs577 Assignment 4 - Report

Girish Rajani-Bathija
A20503736
Department of Computer Science
Illinois Institute of Technology
November 11, 2022

Programming Question 1:

Problem Statement

Download the Kaggle's Cats/Dogs dataset then select a subset of 2000 dogs and 2000 cats while defining training, validation, and testing data generators. A CNN should be built followed by dense layers and flattening the data between the dense and conv layers. The activation of some convolution layers and filters learned should be visualized and analyzed. The custom CNN should be replaced with VGG16 pre-trained model and trained with the conv base frozen, retrained with the conv base unfrozen, and finally trained with the conv base frozen and using data augmentation.

Proposed Solution

The proposed solution is to build several models using different techniques to classify the cats/dogs dataset while also visualizing some activations and filters. The convolutional neural network will contain convolution, pooling, and normalization layers, and the hyperparameters will be tuned. VGG16 will be implemented, trained, and compared with the custom CNN.

Implementation Details

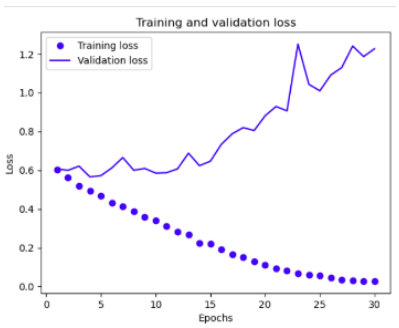
After implementing the CNN and fitting the model, I received the following error: 'UnidentifiedImageError' so I looked at the dataset and found that 1 image from the cat training dataset was corrupted, so I replaced the image with a copy and that solved the problem.

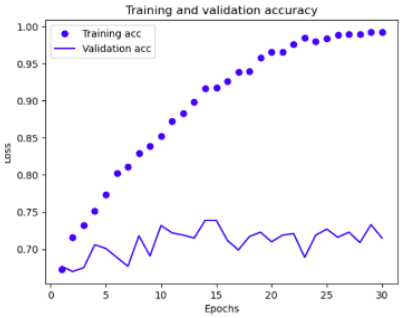
When visualizing the filters learned, the following error occurred: 'tf.gradients is not supported when eager execution is enabled. Use tf.GradientTape instead'. To fix it, I simply disabled the eager-execution constrain from tensorflow by using 'tf.compat.v1.disable_eager_execution()'.

For this assignment and all previous assignments, a dedicated NVIDIA RTX 3070 Ti graphics card with 16GB GPU memory was used to train all models, including VGG16.

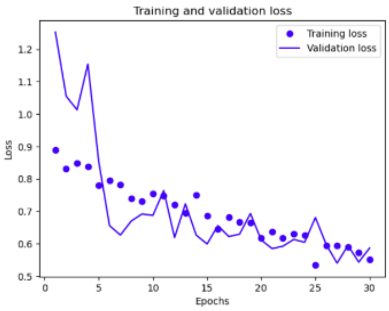
Results and Discussion

From the cats and dogs dataset, a subset of 1000 training samples, 500 validation samples, and 500 testing samples were created for both cats and dogs. The model built consisted of 4 Conv2D layers (the first layer having 32 units, 2nd having 64, and the last 2 having 128 units) with relu activation. In between each conv layer, there was a max pooling and batch normalization. The data was then flattened and passed through 2 dense layers (1 containing 512 units with relu and the output layer containing 1 unit with sigmoid). The RMSprop optimizer with a learning rate of 1e-4 was used with a binary crossentropy loss. Training, validation, and testing generators were created using ImageDataGenerator. The model was trained for 30 epochs with a steps per epoch of 100 and validation steps of 50. The results can be seen below:



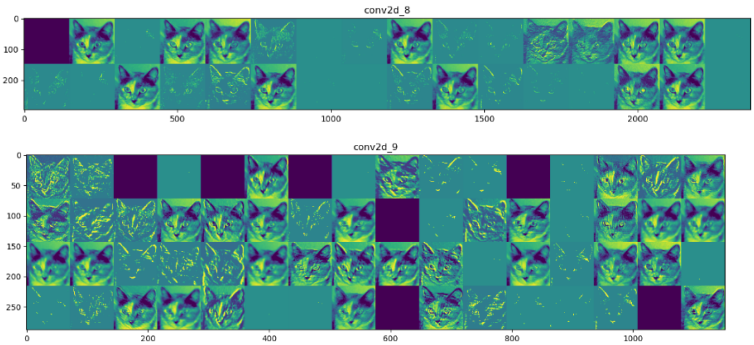


We can see from above that the model starts to overfit after 5 epochs, so to combat this, firstly, a dropout layer of 0.5 was added after flattening the dataset, but that did not make much of a difference, so I added data augmentation to the train data generator by following the lecture implementation and this fixed the overfitting problem. The results are shown below:



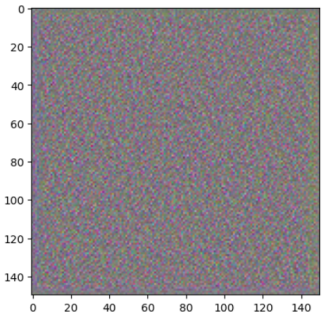

```
Test loss: 0.6522034406661987
Test accuracy: 0.753000020980835
```

As we can see, the model performs significantly better after tuning. The activation in some of the layers was visualized. The activation of the first and fourth layers can be seen below:

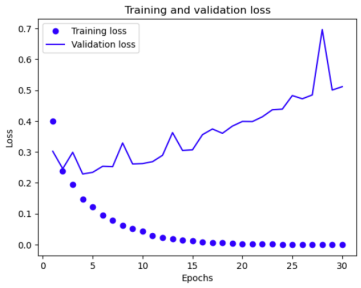


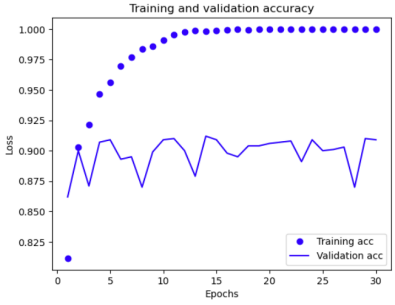
We can see that the early layer (layer 1) has some channels that learn some simple patterns on the cat's face, such as the ears, but in the deeper layer (layer 4), more advanced patterns are identified such as the mouth and face, and we can see that the deeper layer has more activation.

One of the filters in the first convolution layer was vizualized to see what patterns the layer is learning. The following was observed:



The convolution layers in my model were replaced with the pre-trained VGG16 model. To give the correct output size and activation, I added the loaded model as a layer then added a flatten and 2 dense layers with the last dense layer having a unit size of 1 with sigmoid activation. The same optimizer (RMSprop) and loss (binary cross entropy) was used. The conv base was frozen and trained, the results can be seen below:

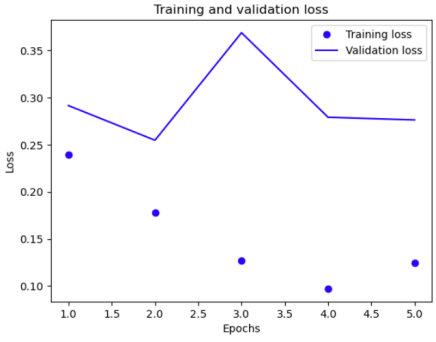


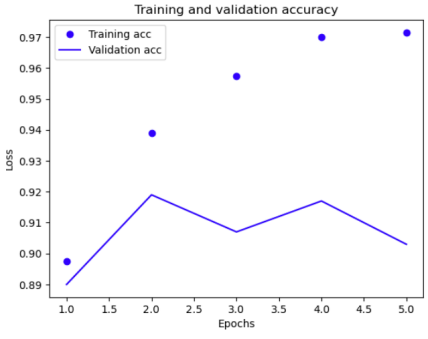



```
Test loss: 0.5693712685722858
Test accuracy: 0.889
```

No data augmentation was applied nor was any dropout or batch normalization added. We can see that the model starts to overfit after appx 3 epochs. However, even after overfitting, the model was still able to get an accuracy of 89% with a loss of 0.57 when evaluated on test data.

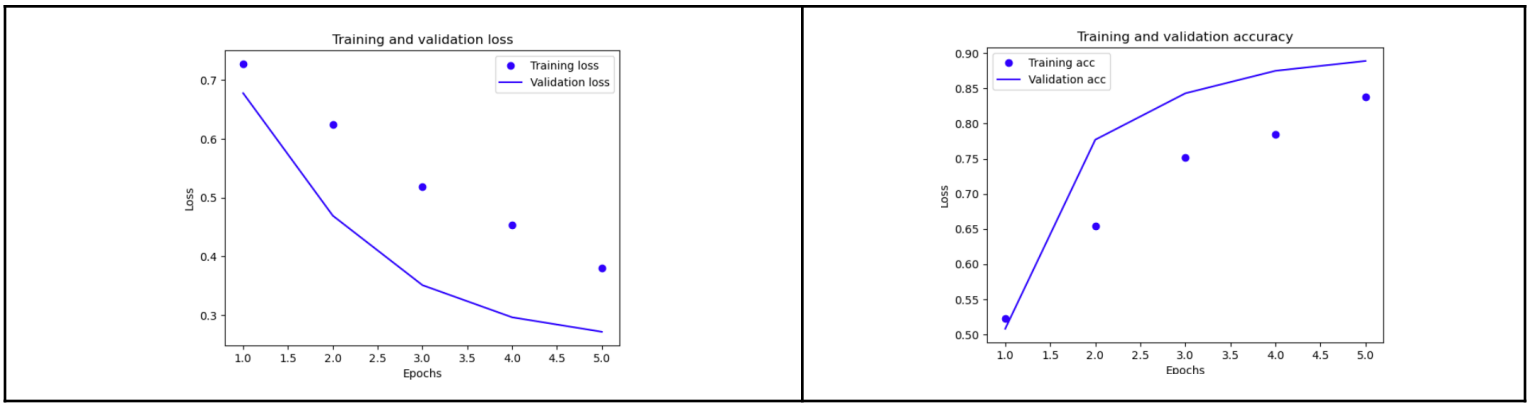
For the next test, I unfroze the conv base and only trained for 5 epochs this time:






```
Test loss: 0.2783838370628655
Test accuracy: 0.903
```

After unfreezing the conv_base and only training for 5 epochs, I was able to get a test accuracy of 90% and a much lower test loss compared to training for 30 epochs with a frozen conv base. For the final model, I modified the data generator to perform data augmentation and retrained it while freezing the conv base. The train data generator with data augmentation contained rescale=1./255, rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True, fill_mode='nearest'. The results:




```
Test loss: 0.2962006203830242
Test accuracy: 0.887
```

From the above results, we can see that the model does not overfit. The validation loss keeps decreasing and the validation accuracy keeps increasing. So, although when evaluated on test data, the results are very similar to the previous model without data augmentation, after adding data augmentation, we completely remove overfitting and get a model that generalizes better.

Programming Question 2:

Problem Statement

Download and load the CIFAR 10 pickled data, vectorize it, and split it into train/validation/test sets. Build convolutional neural networks with multiple convolutions, pooling, and normalization layers while flattening the output and passing it through a single dense layer, then test performance. Add one/two inception blocks, then retest performance, and lastly, replace the inception block with a residual block and evaluate performance on the models built.

Proposed Solution

The proposed solution is to build several convolutional neural networks and testing the performance. The first model will contain a CNN with several convolutions, pooling, and normalization which will be flattened then passed to a single dense layer with a softmax activation. The model will be tuned to deal with overfitting if it occurs. The 2nd model will be built from the first model while introducing an inception block to see how performance improves. The third model will also be built from the first model but instead of using an inception block, a residual block will be used. The results of these 3 models will be analyzed.

Implementation Details

When implementing the inception block, the following error occurred:

```
ValueError: Exception encountered when calling layer "conv2d_116" (type Conv2D).

Negative dimension size caused by subtracting 3 from 2 for '{{node conv2d_116/Conv2D}} = Conv2D[T=DT_
FLOAT, data_format="NHWC", dilations=[1, 1, 1, 1], explicit_paddings=[], padding="VALID", strides=[1,
1, 1, 1], use_cudnn_on_gpu=true](Placeholder, conv2d_116/Conv2D/ReadVariableOp)' with input shapes:
[?,2,2,64], [3,3,64,64].

Call arguments received by layer "conv2d_116" (type Conv2D):
  • inputs=tf.Tensor(shape=(None, 2, 2, 64), dtype=float32)
```

After conducting some research and looking at the implementation from the lecture recordings, I decided to add padding='same' in each convolution layer so that the output size is the same as the input size which would allow for further computation.

When implementing the residual block, the following error occurred:

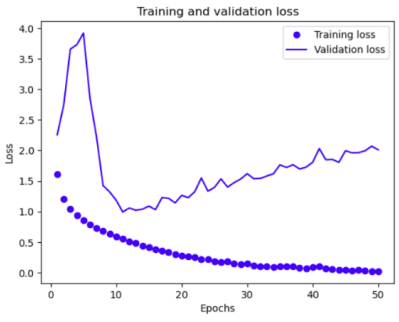
```
ValueError: Inputs have incompatible shapes. Received shapes (4, 4, 64) and (16, 16, 32)
```

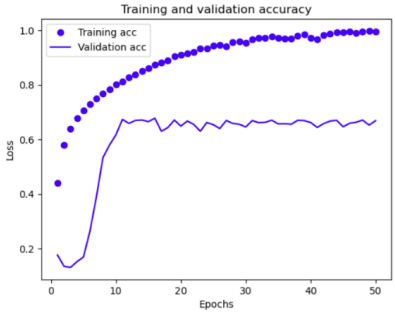
Since the feature map sizes were different, the residual tensor could not be added back to the output features. To make sure that both shapes were the same, I changed the input size of the residual tensor to 64 and added a max pooling layer. This made the residual tensor shape (8, 8, 64) but since we want (4, 4, 64), I added another max pooling layer which then made both shapes compatible with each other and the residual block was created.

Results and Discussion

When working with the CIFAR10 dataset, the dataset was split into validation, training, and testing data. The training dataset consisted of 50,000 samples (40,000 for training, and 10,000 for validation) and the testing dataset contained 10,000 samples. The dataset was vectorized by converting it to float32 and then divided by 255.0. The labels were vectorized using categorical encoding.

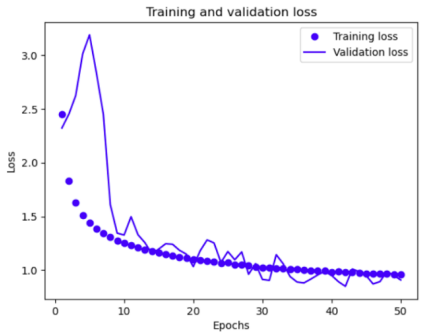
When building the model, various layers and hyperparameters were used to classify the cifar10 dataset. 3 Conv2D layers with relu activation were used. The first Conv2D layer had 32 units per layer while the other 2 had 64. In between each Conv2D layer, batch normalization and max pooling with a pool size of (2,2) were used. Adam optimizer with a lr of 0.001 and categorical crossentropy loss was used. The model was trained for 50 epochs with a batch size of 512.

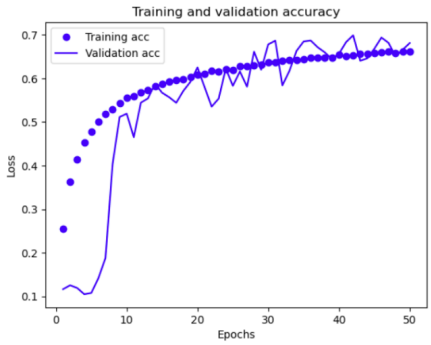





```
Test loss: 2.070465564727783
Test accuracy: 0.6635000109672546
```

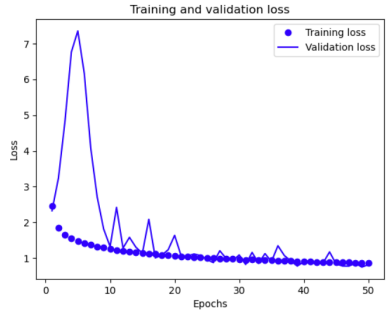
From the results shown above, we can see that the model starts to overfit after 10 epochs and when evaluated on test data, got a loss of 2.07 and an accuracy of 66.35% which is not good. To reduce overfitting and decrease the loss, I added a dropout of 0.5 after each convolution layer. The results of this tuning can be seen below.

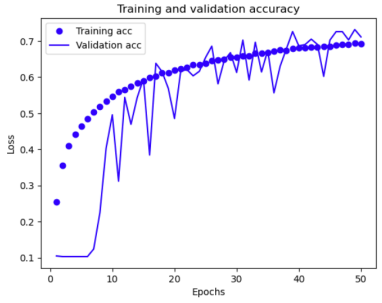





```
Test loss: 0.9167814254760742
Test accuracy: 0.6782000064849854
```

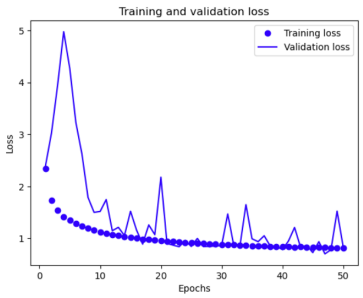
As shown above, we can see the model does not overfit now and the loss has significantly reduced. I had also tried tuning other parameters such as adjusting the learning rate in the Adam optimizer and also trying SGD optimizer but neither of those solutions had any positive impact. When implementing the inception block, I kept a very similar format for the model but just added a few additional layers with max pooling, batch normalization and dropout. The results of the inception block implementation can be seen below:






```
Test loss: 0.8352344036102295
Test accuracy: 0.7046999931335449
```

As shown above, after implementing the inception block, the result are a bit better, the model performs well while have a lower loss and slightly higher accuracy when evaluated on test data compared to the previous model without the inception block. A third model was created now which has the same model architecture as the first, meaning that it has the same Conv2D layers, max pooling, batch normalization, and dropout. The results of this implementation can be seen below:




```
Test loss: 0.850620687007904
Test accuracy: 0.7006999850273132
```

The residual block implementation performed very similarly to the inception block based on the results shown above. In the end, implementing either an inception block or residual block was able to slightly increase the accuracy of the model and lower the loss.