ML2 Final Project

Aaron Skipper and Yu Xi

GROUP 7- NATURE DATASET – CNN CLASSIFICATION

APRIL 29, 2019

Outline

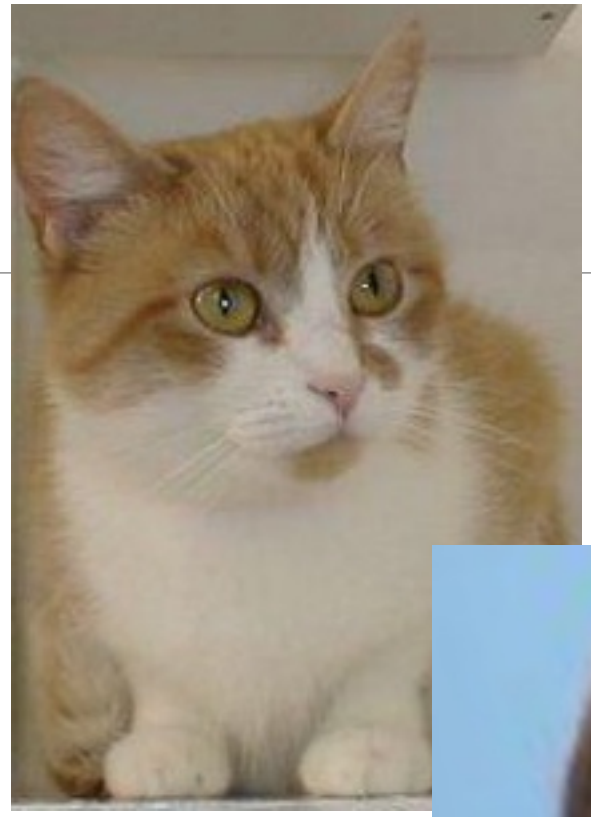
- Introduction
- Motivation
- About the data
- Problem statement
- Pre-processing
- Model Explanation
- Results
- Conclusion

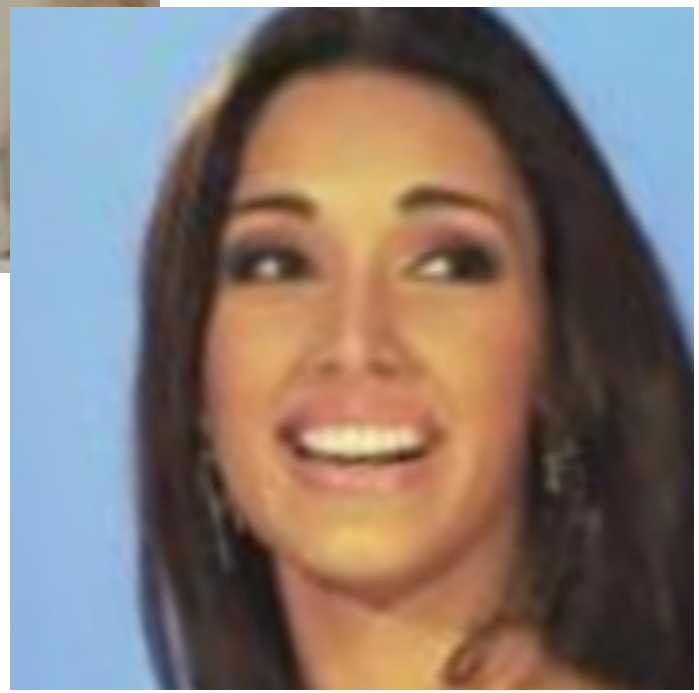
Motivation / About the data

- To further practice with image classification which represent a variety of classification problems within large population of professional industries.
- Nature dataset was used and down loaded from Kaggle: https://www.kaggle.com/prasunroy/natural-images/kernels

- Dataset consist of 6899 images of 8 classes
(airplane, car, cat, dog, motorbike, person, flower, fruit)

Images





Problem statement

To create a CNN network that will perform to correctly classify the 8 classes within 6899 images of the natural dataset.

- We will use the PyTorch framework to create a convolutional network (CNN) to apply to the natural dataset.

Pre-processing- Partition

- The dataset will be partitioned into 3 sub-datasets, using the dataset method from torchvision:

```
from torchvision import transforms, datasets
```

  - Train (80%)
  - validation (10%) – used to provide unbiased evaluation of the model and tune the hyperparameters
  - test(10%) – used to evaluate the final model chosen.

Pre-processing- Partition

Results of sizes of the partitioned data

```
all_data = datasets.ImageFolder(root='./natural_images')
train_data_len = int(len(all_data)*0.8)
valid_data_len = int((len(all_data) - train_data_len)/2)
test_data_len = int(len(all_data) - train_data_len - valid_data_len)
train_data, val_data, test_data = random_split(all_data, [train_data_len, valid_data_len, test_data_len])
```

```
print(len(train_data), len(val_data), len(test_data))
```

```
5519 690 690
```

Pre-processing - transform

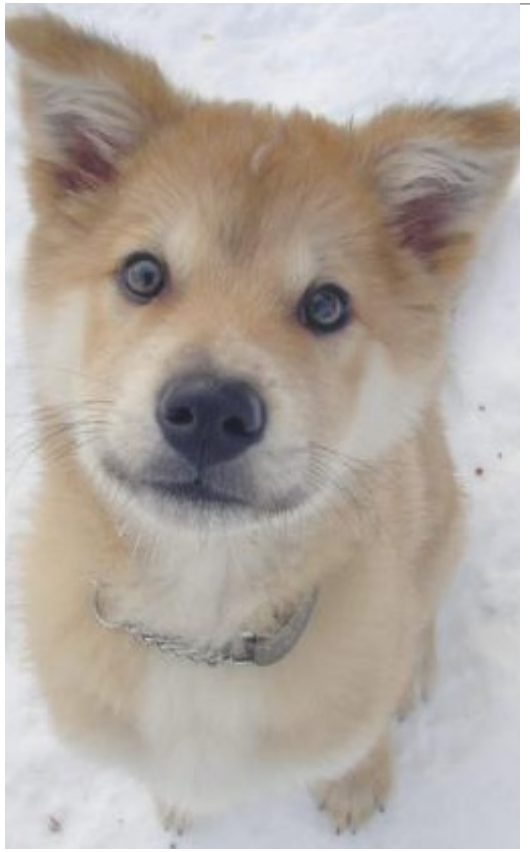
- Using the transform method from torchvision, the training dataset is augmented with different representations of the images
  - Random Rotation
  - Random horizontal flip
  - Resized crop
  - Color jitter

The images then were normalized and the datatype was changed to torch tensors, to be able to apply the images within the network, represented in pytorch.

Pre-processing- transform

```
image_transforms = {
    # Train uses data augmentation
    'train':
    transforms.Compose([
        transforms.RandomResizedCrop(size=256, scale=(0.95, 1.0)),
        transforms.RandomRotation(degrees=15),
        transforms.ColorJitter(),
        transforms.RandomHorizontalFlip(),
        transforms.CenterCrop(size=224),  # Image net standards
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])  # Imagenet standards
    ]),
    'val':
    transforms.Compose([
        transforms.Resize(size=256),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test':
    transforms.Compose([
        transforms.Resize(size=256),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

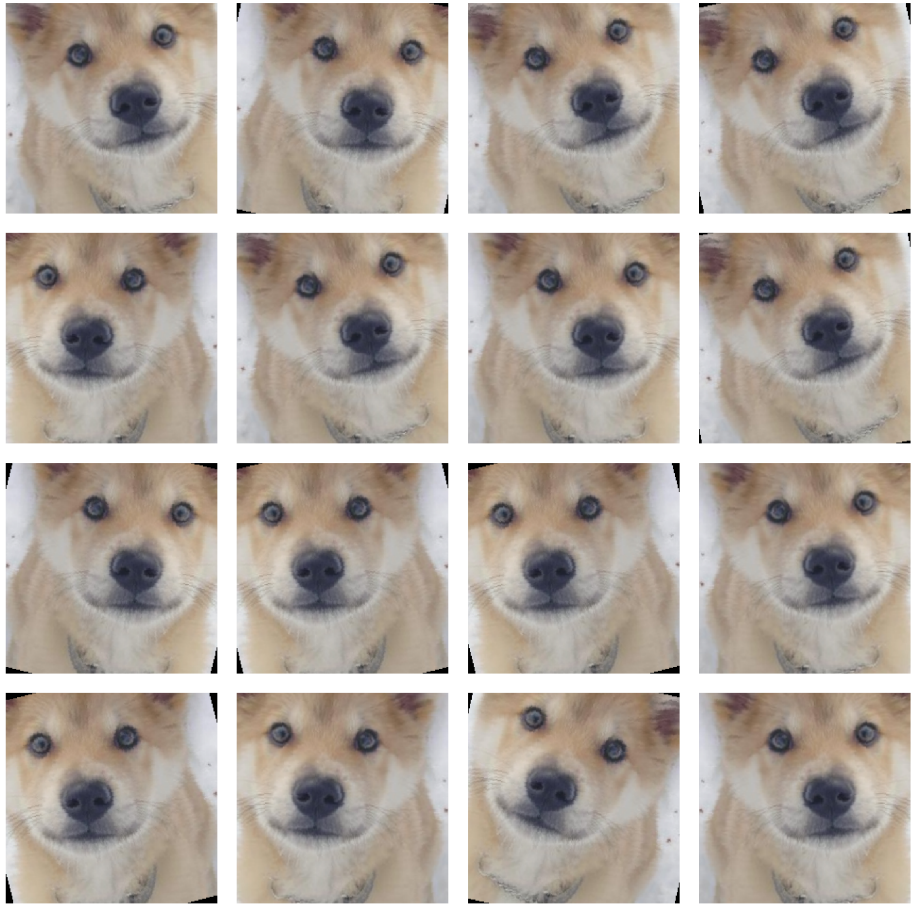
Pre-processing- transform



The left is a random image of a dog within the dataset.

The right shows 16 possible ways that the image can be randomly transformed

Doing this enables the network to be more robust within its training



Model Structure

- Convolution neural network was utilized to address the classification of the natural dataset and it’s 8 classes
- CNN networks are widely used for image analysis as they work well with making sense of patterns within images

- Our basic model structure consisted of 2 convolutional layers

Model Explanation

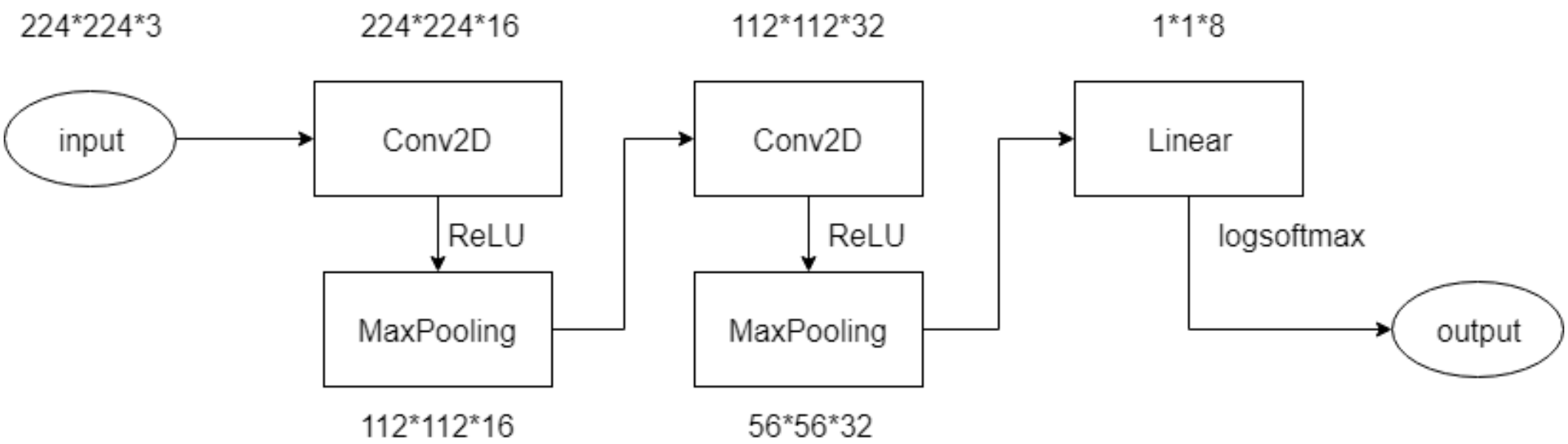
- The VGG-16 pre-trained model was referred to guide the selection select of the hyperparameters, along with the module setup-
  - Batch size 128
  - Transfer function- ReLU
  - Kernel size (3x3)
  - Stride: 2
  - Padding (1,1)
- As this is a classification problem, cross entropy /loss function and model accuracy were used to analyze the performance of the models.
- ADAM optimizer was applied to attempted models as our research and experience shows the it works best

Hyperparameters for performance analysis

Hyperparameters that were experienced with to analyze model performance:

```
#-----------------------------------------
# Hyperparameters Defined
#-----------------------------------------
batch_size = 128
num_epochs = 5
learning_rate =1e-3
#-----------------------------------------------------------------------------------------------------------------------
```

Model Architecture- 2 layer CNN



Model Structure 2-layer CNN/ Basic-Code

```
# Class for 2 convolutinal - layer network
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(3, 3), padding=(1, 1)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False))
        self.fc = nn.Linear(56 * 56 * 32, 8)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        out = self.softmax(out)
        return out
```

Model Explanation (basic)

Obtained accuracy of 89% with using 2 layer convolutional network. This model was used as our basic model, from where we would apply techniques to improve its performance.

- ADAM optimizer was applied. From research and experience we found this to be the best optimizer to apply

airplane	727.0	93.155433	19.365537	54.0	...	290.50	296.0	303.00	383.0
car	968.0	100.000000	0.000000	100.0	...	100.00	100.0	100.00	100.0
cat	885.0	303.732203	95.744543	73.0	...	234.00	315.0	405.00	499.0
dog	702.0	311.933048	99.654237	50.0	...	204.25	274.0	366.75	495.0
flower	843.0	328.167260	189.871326	59.0	...	225.00	365.0	485.00	2737.0
fruit	1000.0	100.000000	0.000000	100.0	...	100.00	100.0	100.00	100.0
motorbike	788.0	109.114213	12.083767	66.0	...	186.00	193.0	198.00	213.0
person	986.0	255.981744	0.330622	250.0	...	256.00	256.0	256.00	256.0

[8 rows x 16 columns]
5519 690 690
torch.Size([128, 3, 224, 224]) torch.Size([128])
Test Accuracy of the model on the 690 test images: 89 %

Model Optimizations

- To optimize the performance of our base 2-layer CNN model, we experimented with:
  - Hyperparamters
    - Number of epochs
    - Learning rate
  - Added convolutional layers
  - Dropout rate

Optimization Scheme

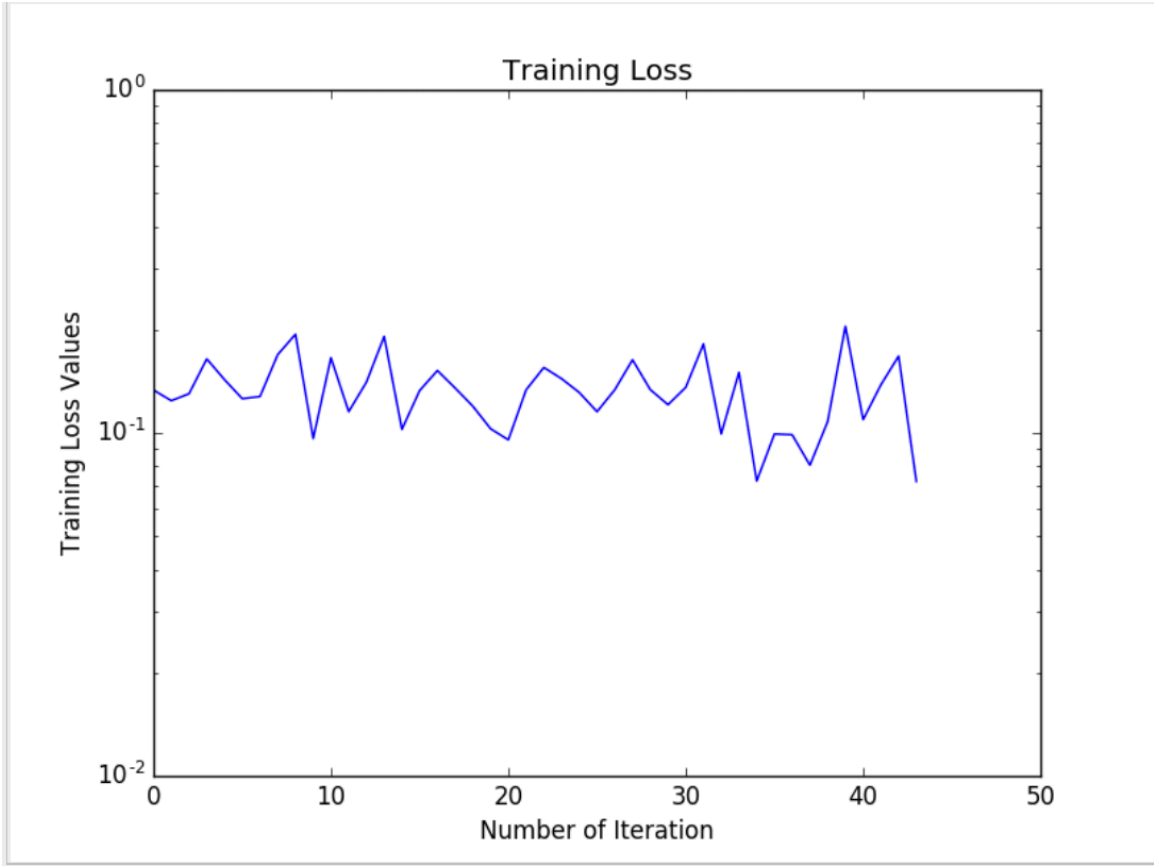
- Keeping the layer number and other parameters of the basic model, different learning rates were applied to evaluate its influence on the performance.
- If the accuracy on the training dataset is low, we will increase the number of epochs.
- If there is a large difference between accuracies of training and validation dataset, we will decrease the number of epochs to prevent overfitting.
- Once the choose the best model, we will introduce the dropout rate to the model and find the best one.

Results- 2 CNN w/ different learning rates

Accuracy of a 2-layer CNN on training and validation dataset

Learning Rate	Training Accuracy (%)	Validation Accuracy (%)
1e-02	91.4	83.8
1e-03	98.6	90.2
1e-04	98.4	66.7

Confusion matrix and Training loss for 2-layer CNN



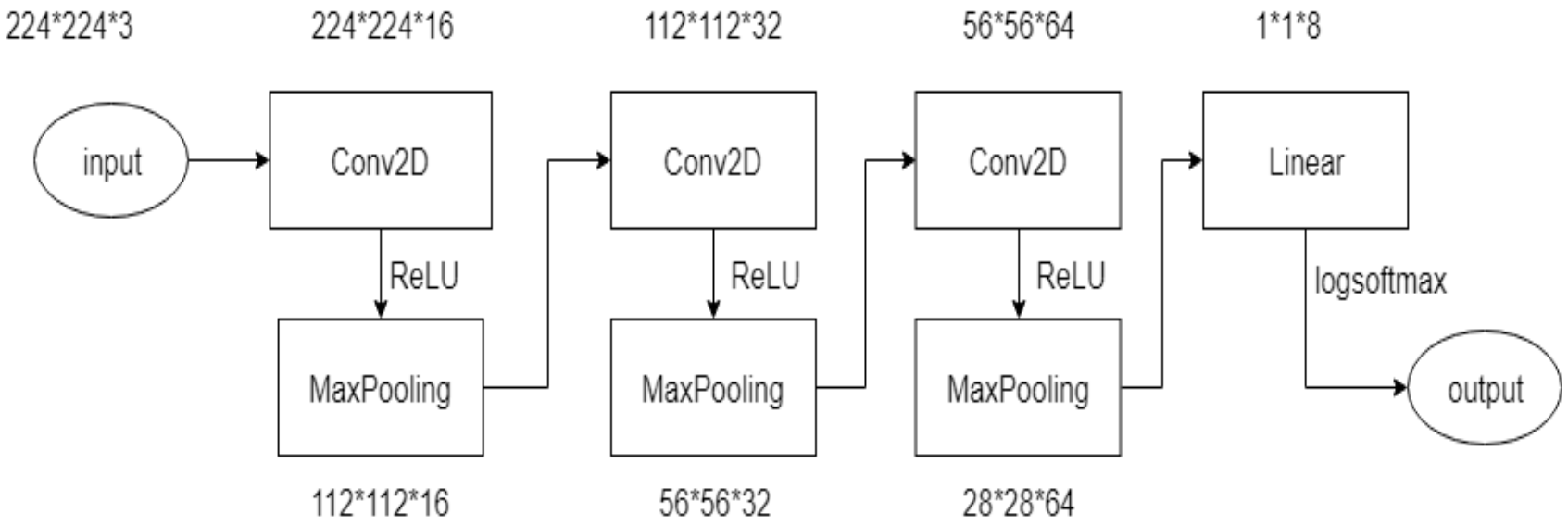
classes					...	
airplane	727.0	93.155433	19.365537	54.0	...	2
car	968.0	100.000000	0.000000	100.0	...	1
cat	885.0	303.732203	95.744543	73.0	...	2
dog	702.0	311.933048	99.654237	50.0	...	2
flower	843.0	328.167260	189.871326	59.0	...	2
fruit	1000.0	100.000000	0.000000	100.0	...	1
motorbike	788.0	109.114213	12.083767	66.0	...	1
person	986.0	255.981744	0.330622	250.0	...	2

[8 rows x 16 columns]
5519 690 690
torch.Size([128, 3, 224, 224]) torch.Size([128])
Test Accuracy of the model on the 690 test images: 90 %

Predicted	0	1	2	3	4	5	6	7	All
Actual									
0	4	0	0	0	0	0	0	0	4
1	0	6	0	1	0	0	0	0	7
2	0	0	2	1	0	0	0	0	3
3	0	0	1	4	0	0	0	0	5
4	0	0	0	0	11	0	0	0	11
5	0	0	0	0	0	10	0	0	10
6	0	1	0	0	0	0	2	0	3
7	0	0	0	0	0	0	0	7	7
All	4	7	3	6	11	10	2	7	50

3 layer CNN

- 1 layer was added to basic model for a total of 3 convolution layers
  - The training accuracy is 98.8%, which is close to the 2-layer network. However, the validation accuracy is 88.4%, which is worse than the performance of the 2-layer CNN.



4 layer CNN

- Adding another layer gives a 4-layer CNN.
- We also experiment on a variant of that using only 2 max pooling layers instead of 4.
- For the first trial using 5 epochs, the accuracy on the training set is low, which means the model is underfitting the data.
- We then increased the number of epochs to 8.
- As a result, the training accuracy increased but the validation accuracy decreased.
- The variant of the 4-layer CNN performs worse on the validation dataset than other models

4 layer CNN results / Architechture

Number of Max Pooling Layers	Number of epochs	Training Accuracy (%)	Validation Accuracy (%)
4	5	92.8	88.7
4	8	96.1	87.7
2 (variant)	5	98.5	86.2

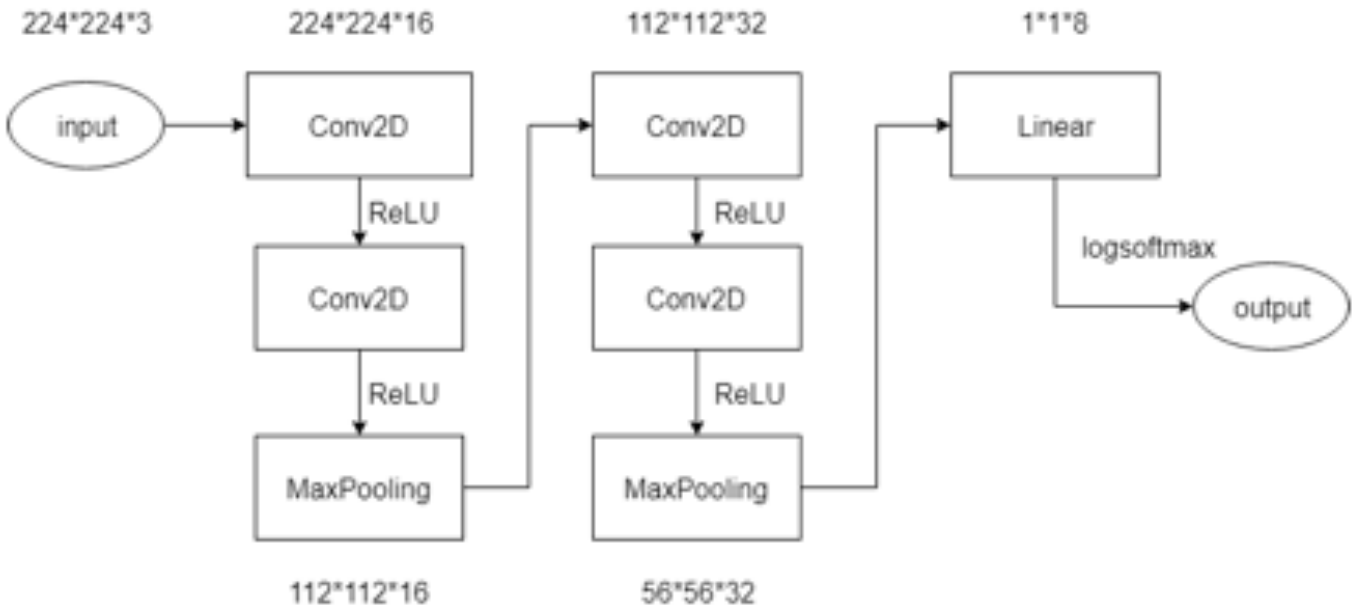
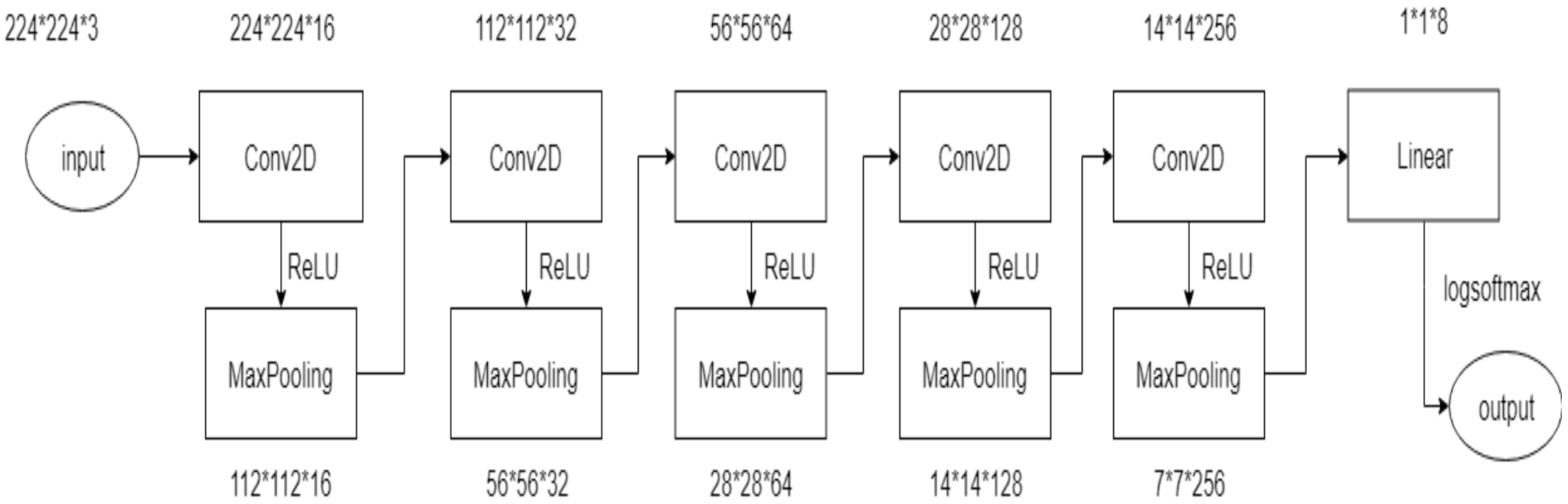


Figure 4-2. Architecture of a 4-layer CNN (Variant)

5 layer CNN

- Without using dropout layer, the model gives the best performance so far with a validation accuracy of 91.4% using 8 epochs.
- Since it is the model gives the highest validation accuracy so far. We tested on using an additional dropout layer on the model.
- The two models using a dropout layer does not perform so well as without using it.

5 layer CNN Architecture



5 layer CNN Experimental results

Number of epochs	Dropout Rate	Training Accuracy (%)	Validation Accuracy (%)
5	0	92.0	89.1
8	0	97.6	91.4
10	0	95.3	86.7
8	0.2	99.5	89.9
8	0.4	95.6	90.9

6 layer CNN

- Lastly, we tried to experiment on a 6-layer CNN.
- Adding another convolution and max pooling after 5-layer CNN does not make much sense, because the dimension of the output would be too small.
- Using a structure similar to Figure 4-2 with 6 convolution layers and 3 max pooling layers will run out of GPU memory.

After choosing the best model, we use test dataset to provide an unbiased evaluation on the 5-layer model. The test accuracy is 91.2%, which is close to the validation accuracy as expected.

Parameters evaluated for influencing performance

Epoch, learning rate, # of convolutional layers, dropout

Accuracies of all CNN models on training and validation dataset

Number of Convolution Layer	Number of Max Pooling Layer	Learning Rate	Dropout Rate	Number of epochs	Training Accuracy	Validation Accuracy
2	2	1e-02	0	5	91.4	83.8
2	2	1e-03	0	5	98.6	90.2
2	2	1e-04	0	5	98.4	66.7
3	3	1e-03	0	5	98.2	88.4
4	4	1e-03	0	5	92.8	88.7
4	4	1e-03	0	8	96.1	87.7
4	2	1e-03	0	5	98.5	86.2
5	5	1e-03	0	5	92.0	89.1
5	5	1e-03	0	8	97.6	91.4
5	5	1e-03	0	10	95.3	86.7
5	5	1e-03	0.2	8	99.5	89.9
5	5	1e-03	0.4	8	95.6	90.9

Summary / Conclusion

- The best model we have is a 5-layer CNN with learning rate = 1e-03, number of epochs = 8, and dropout rate = 0.
  - **The test accuracy of this model is 91.2%.**

- With the hyperparameters held constant, a CNN model with more layers does not necessarily perform better. Also, using increased number epochs does not always increase the accuracy on the training dataset.
- In our experiments, using a dropout layer improves the training accuracy, but not for the validation accuracy. Changing other parameters along with the dropout layer may give a better model.

Going forward

- One of the improvements that could be made, would be to do more experimenting with the parameters that were not tested. Ie. Batch size..
- The combinations of the parameters that were not tested may provide more information on the effects.
- If we had more GPU memory, we could test the model with more convolution layers and different architecture of networks
- With increasing of convolution layers experimented with, using BatchNorm method from torchvision addresses covariate shift phenomena in CNN models. It was experimented with within VGG-16 pre-training research.