STAT 542 Midterm Project Presentation

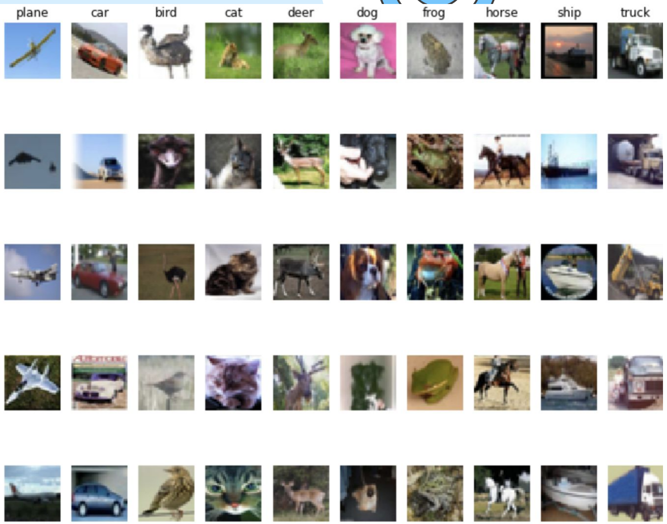
Group 8:

Group Member: Yefan Li, Jue Li, Yijin Wang, Xiao Ma, Wenxuan Gu, Pengru Lyu, Ziyi Xue, Yanqing Li

Dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 10,000 randomly-selected images from each class.



Yanqing Li

Support Vector Machines (SVM)

- SVM is a set of supervised learning methods that find the hyperplanes to separate datasets.

- Pros & Cons of SVM (with this dataset):

  Pros : effective in high dimensional spaces, by mapping input data with different kernels

  Cons:

  a. computationally intensive, especially we choose the k-fold CV method for turning parameters

  b.highly affected by the choice of the kernel

Ziyi Xue

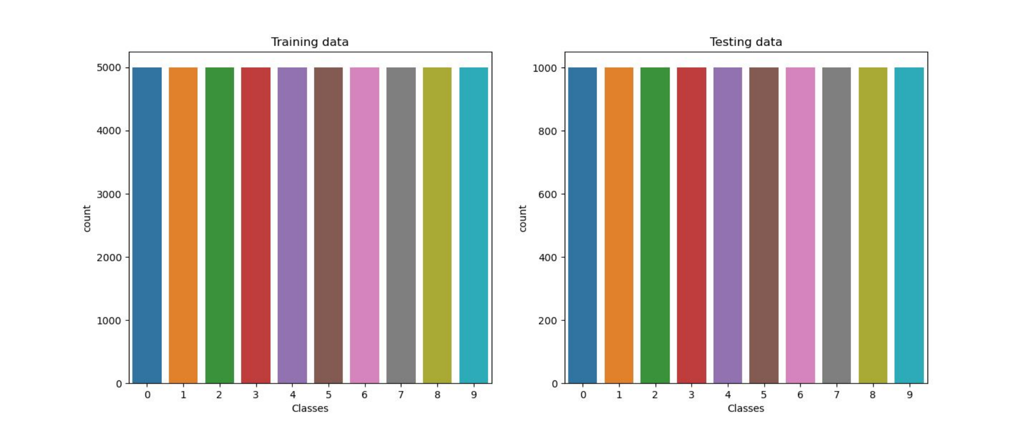
Preprocessing Data

Shape of the dataset :

a. Change 32 * 32*3 color image data into 1 Dimension (potential problems caused by the really high dimensions)

b. Scale the pixel value in [0,1]

Balance of the data:



Ziyi Xue

SVM Model Parameters & Selection

Our SVM model is based on tuning the kernels and the parameters c and gamma:

```
finalModel = SVC(C=1, gamma=`scale`, kernel='poly')
```

- Parameters:

  Regularization parameter (c) :

  The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

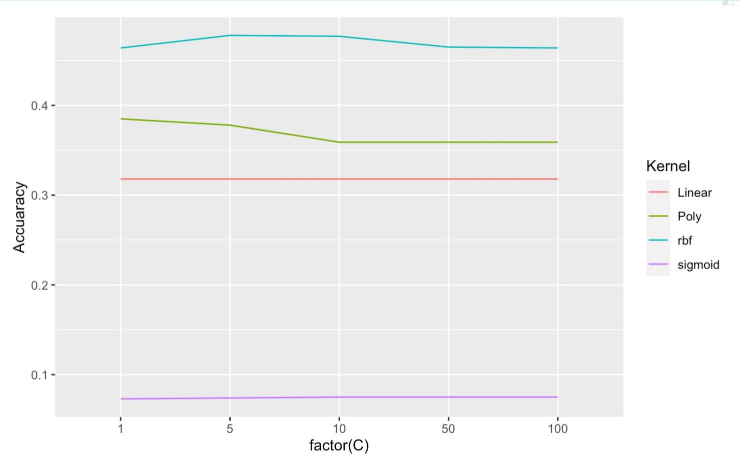
  gamma: determine the width of the kernel (useless for linear kernel)

  Kernel: 'linear'; ' poly'; 'rdf'; 'sigmoid'

  Degree: Degree of the polynomial kernel function ('poly') and ignore by all other kernel.

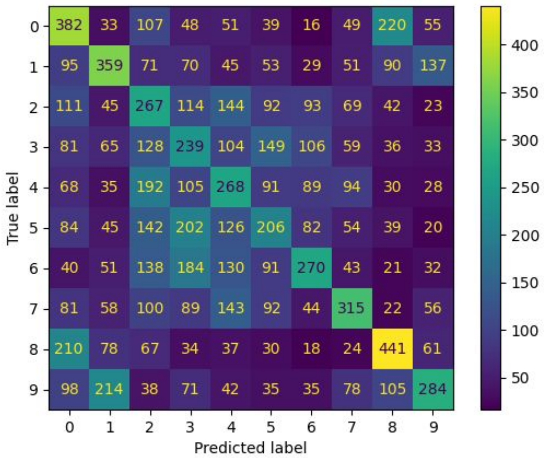
- Method:

  1) Fixed gamma value, different C value (1, 5, 10, 50, 100)
  2) Fixed C value, different gamma value (1e-2, 1e-3, 1e-4)



Jue Li

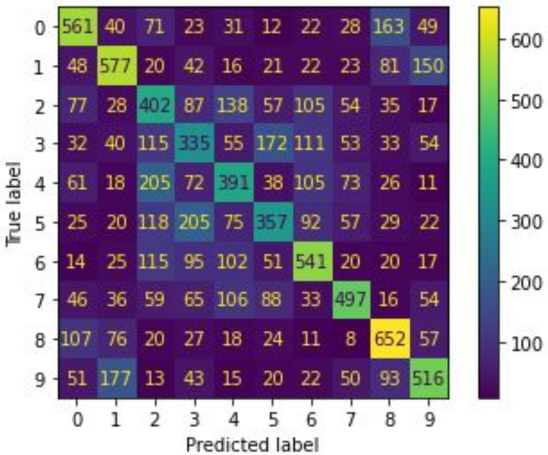
Results & Performance of SVM




```
linear_model = SVC(kernel='linear')
```

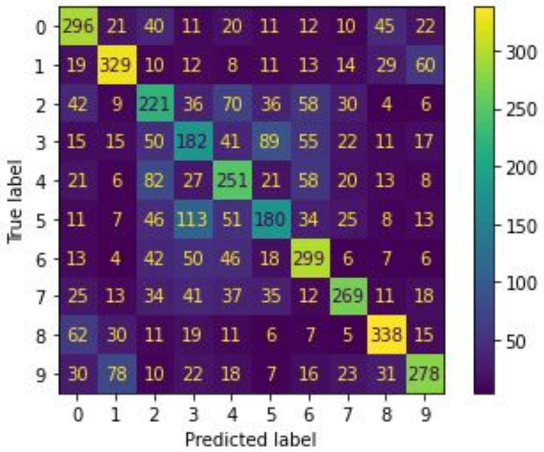
Linear SVM with accuracy around 30.31 %

10000 trained and test data




```
SVC(C=5, gamma='scale', kernel='rbf', degree=2)
```

Nonlinear (rbf) with parameter tuning:
Accuracy: 48.29%
Runtime: 1283.28 s (~21.4 min)




```
SVC(kernel='poly')
```

Nonlinear (poly) without parameter tuning:
Accuracy: 52.86%
Runtime: 9747.61 s (~3hr)

50000 trained data and 5000 test data

Yijin Wang

Difficulty & Future Work...

Difficulties & Problems:

- **Computation complexity:** for high dimensions (p = 3072) and large sample size (for both training and testing data)
- **Noise of dataset:** the images have much noises, which lead to the bad performance of SVM
- **Long runtime:** somewhat hard to run the whole 50,000 dataset at once therefore not having the most accurate result

Possible Solutions:

- **Computation complexity:** parallel processing
- **Noise of dataset:** data smoothing methods, such as penalized models and resample with different ratios.
- **Long runtime:** try to use CPU or GPU to run the dataset more faster

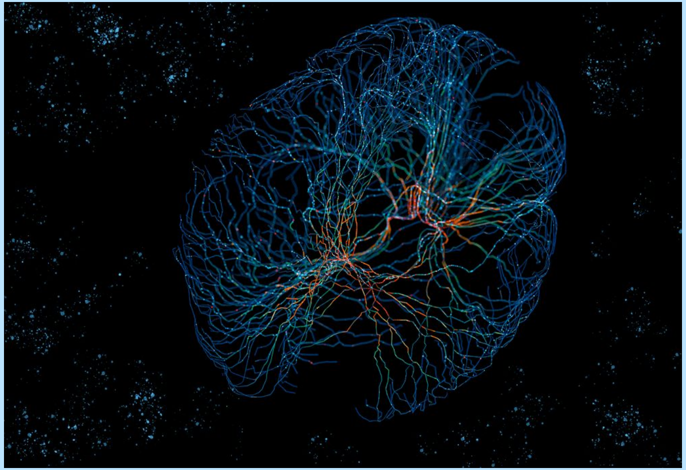
Yijin Wang

Convolutional Neural Network(CNN)

What is neural network?

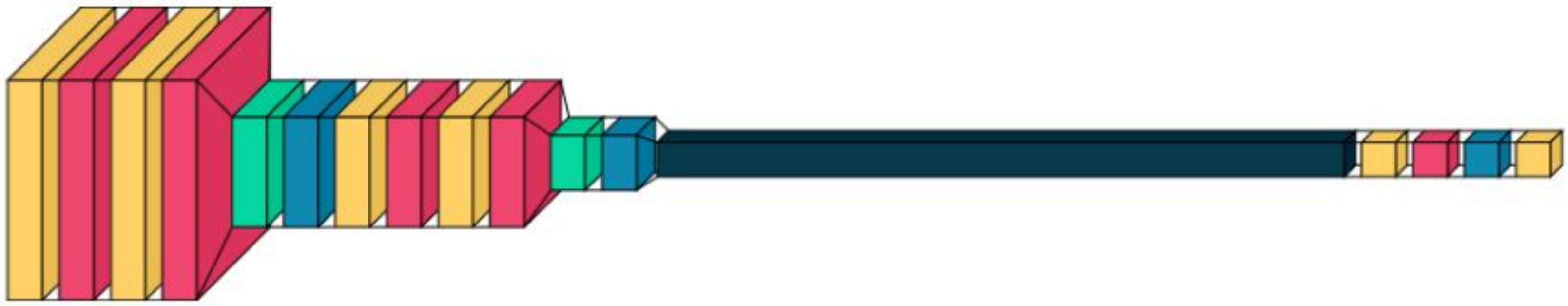
What is convolutional neural network (CNN)?

Why CNN?



Pengru Lyu

Base Model Structure



Our Neural Network Model process can be explained as above

32 - 32 - 64 -64

Pengru Lyu

Activation Function Selection: Sigmoid VS ReLU

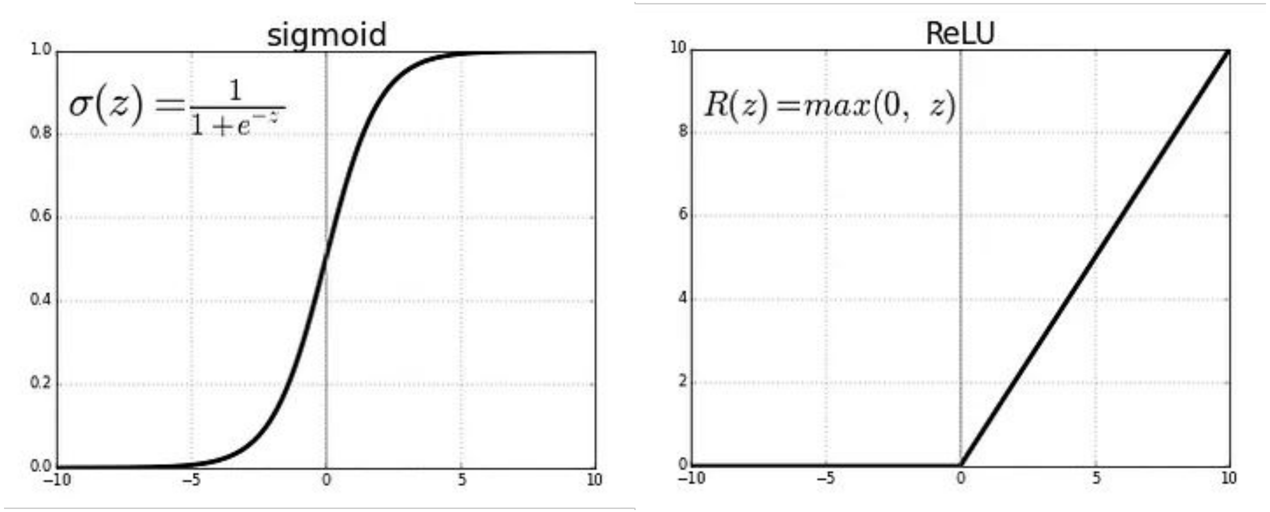


Figure: Sigmoid vs. ReLU

Source: https://miro.medium.com/v2/resize:fit:720/format:webp/1*XxxiA0jJvPrHEJHD4z893g.png

Sigmoid vs. ReLU

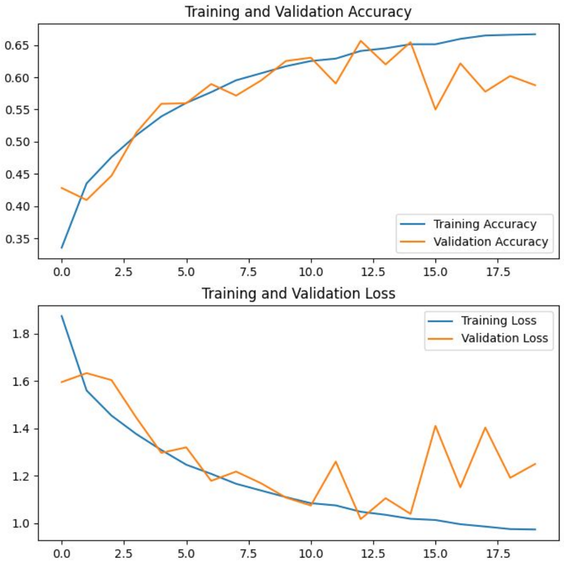


Figure: Sigmoid
Sigmoid Accuracy: 0.5877 (approx.)

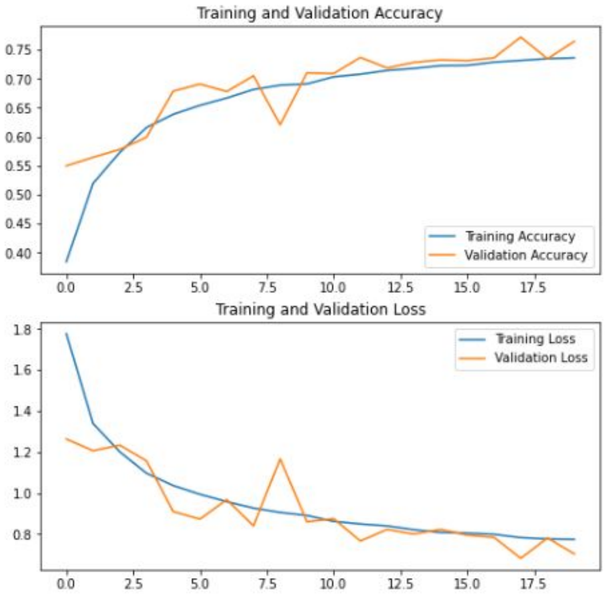


Figure: ReLU, accuracy: 0.7533 (approx.)

Pengru Lyu

First Attempt: Add More Convolutional Layers

32-64-128

```
[4]  amount_layers = len(model.layers)
     amount_layers

     17
```

```
model_summa = model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 64)                262208

 batch_normalization_4 (Batc  (None, 64)               256
 hNormalization)

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 329,450
Trainable params: 328,938
Non-trainable params: 512
```

```
[9]  ayer_amount = len(model.layers)
     ayer_amount

     23
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 32, 32, 32)        896

 batch_normalization_14 (Bat  (None, 32, 32, 32)       128
 chNormalization)

 conv2d_13 (Conv2D)          (None, 32, 32, 32)        9248

 batch_normalization_15 (Bat  (None, 32, 32, 32)       128
 chNormalization)

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 dropout_8 (Dropout)         (None, 16, 16, 32)        0

 conv2d_14 (Conv2D)          (None, 16, 16, 64)        18496

 batch_normalization_16 (Bat  (None, 16, 16, 64)       256
 chNormalization)

 conv2d_15 (Conv2D)          (None, 16, 16, 64)        36928

 batch_normalization_17 (Bat  (None, 16, 16, 64)       256
 chNormalization)

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_9 (Dropout)         (None, 8, 8, 64)          0

 conv2d_16 (Conv2D)          (None, 8, 8, 128)         73856

 batch_normalization_18 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 conv2d_17 (Conv2D)          (None, 8, 8, 128)         147584

 batch_normalization_19 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_10 (Dropout)        (None, 4, 4, 128)         0

 flatten_2 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 128)               262272

 batch_normalization_20 (Bat  (None, 128)              512
 chNormalization)

 dropout_11 (Dropout)        (None, 128)               0

 dense_5 (Dense)             (None, 10)                1290
```

Xiao Ma

Second Attempt: Transfer Learning

- Introduction: Taking a pre-trained neural network and adapting it to a new task
- Popular models: GoogLeNet network, ImageNet
- Results:

```
- val_output_accuracy: 0.2540
```

```
Test accuracy: 0.7008000016212463
```

Third Attempt: Learning Schedule and Regularization

- Learning Schedule: Also known as learning rate schedule or learning rate decay, involves adjusting the learning rate during training

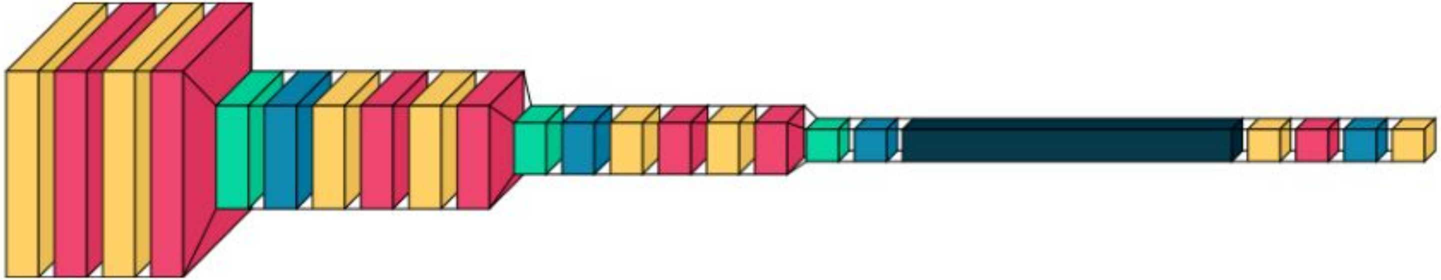
```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=train_images.shape[0] // batch_size,
    decay_rate=0.95,
    staircase=True)
```

- Regularization: Adding additional constraints to the training process to prevent overfitting
  Method: Dropout

```
layers.Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg),
layers.BatchNormalization(),
layers.MaxPooling2D((2, 2)),
layers.Dropout(0.5),
```

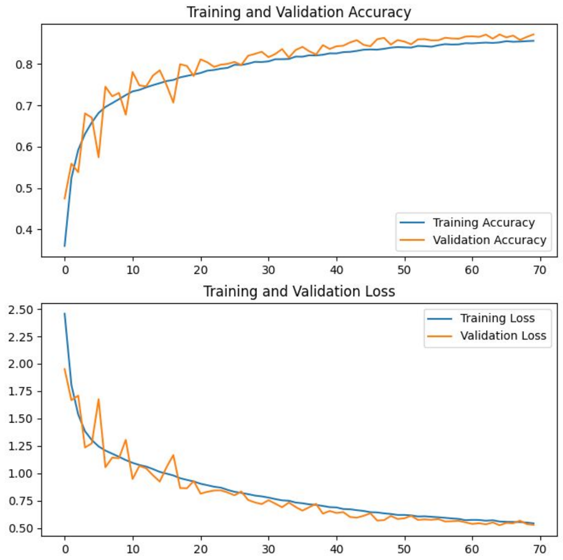
Xiao Ma

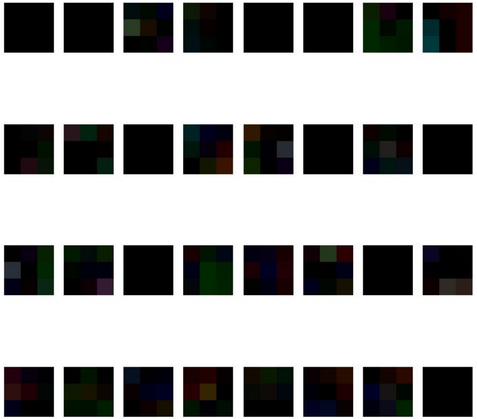
Final Model Structure



Wenxuan Gu

Final Model Accuracy





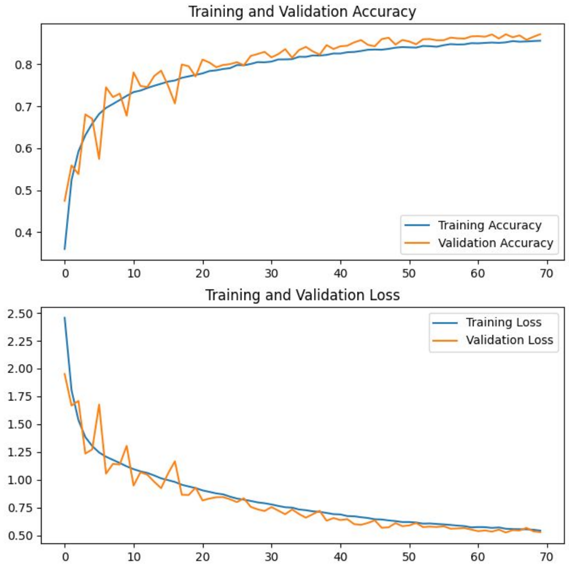
Accuracy: 0.8715999722480774

Wenxuan Gu

Way to Find the Maximum Epochs Needed

1. Pseudocode:

```
Epoch = 1000
Run model
Count == 0
While {
Track every accuracy outputted by Model
If ( Accuracy_{i} - Accuracy_{i-1} <= N )
Count += 1
If (count == 3)
Terminate model
Break}
```



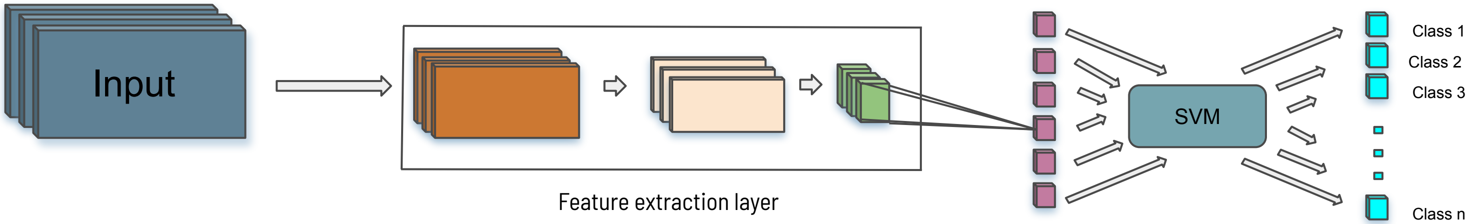
2. Example:

```
Epoch 11/50
781/781 [==============================] - 532s 681ms/step - loss: 1.1138 - accuracy: 0.7327 - val_loss: 1.0703 - val_accuracy: 0.7528
Epoch 12/50
781/781 [==============================] - 511s 655ms/step - loss: 1.0839 - accuracy: 0.7361 - val_loss: 0.9837 - val_accuracy: 0.7686
```

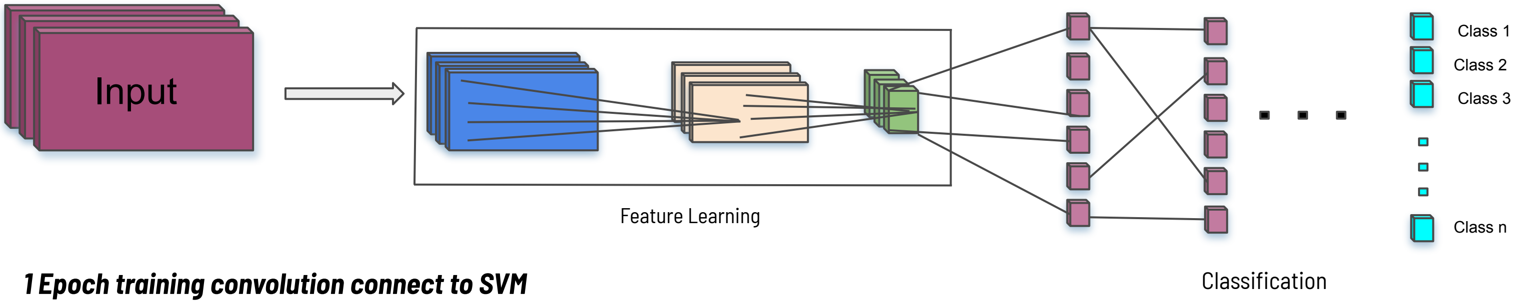
Wenxuan Gu

Convolutional SVM

We used CNN as a socket for SVM, and the flowchart would be shown as below:



CNN Flowchart:



1 Epoch training convolution connect to SVM

CNN accuracy:0.4151

Convolutional SVM accuracy:0.5797

Wenxuan Gu

Conlusion

Performance (SVM vs. CNN)

Why ? (SVM suited for simpler datasets with fewer dimensions, CNN works better for image recognition task)

Thanks!

Do you have any questions?