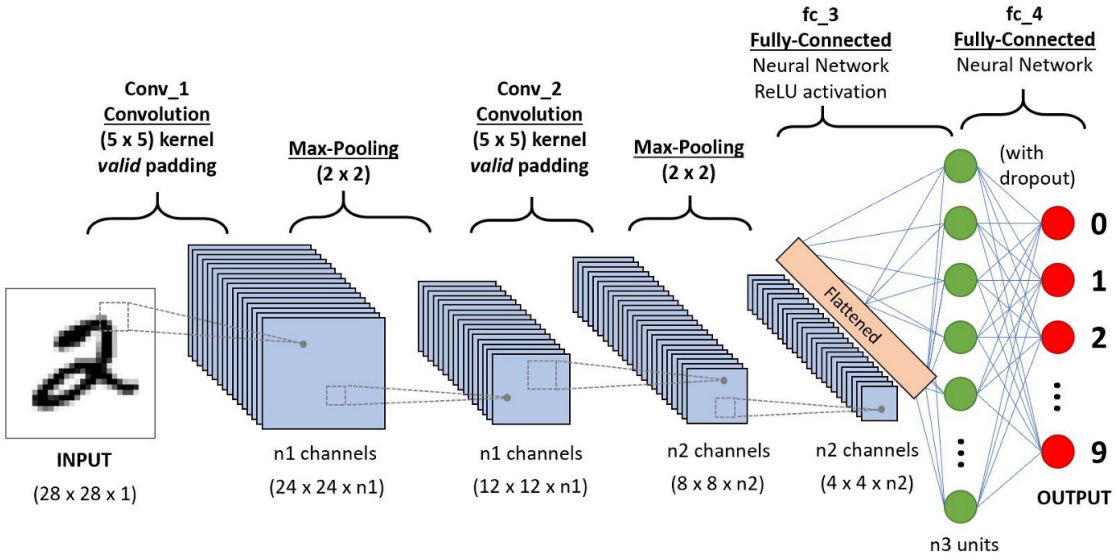
Classification of Street View House Numbers (SVHN) dataset Using CNN

➢ Detailed explanation of how CNN works (if necessary, include visualization)
➢ Explanation of each and every parameter/hyperparameter of CNN with its function.

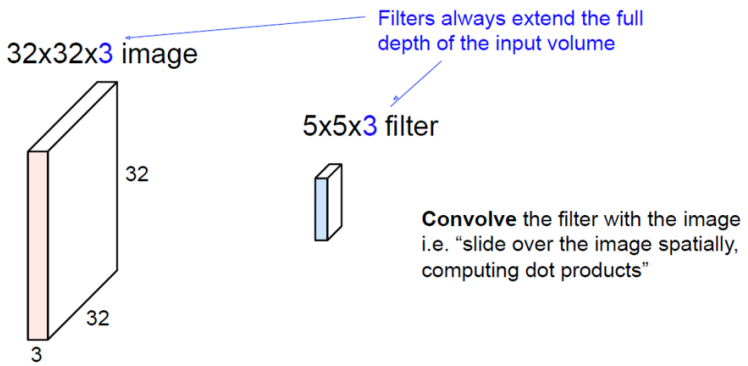
In this part, I will explain the basic concept of how CNN works and its parameter/hyperparameters.



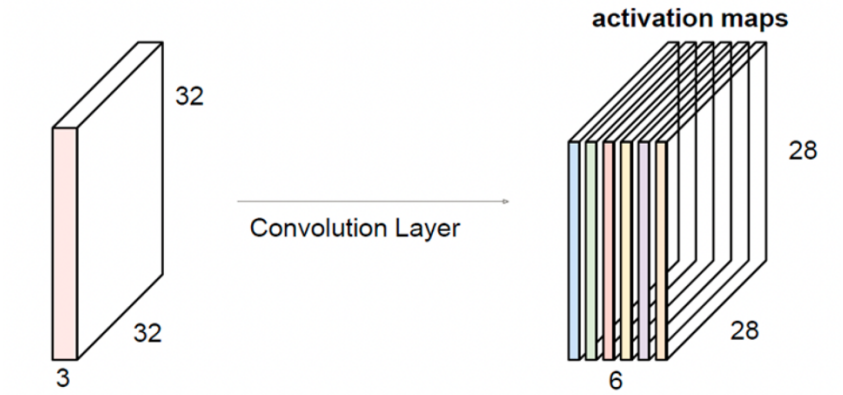
CNN typically has 3 components (layers)

1. **Convolutional layer**
   - Creates a feature/activation map by first applying convolutional filters to the image and an activation function to the filtered image to make the model non-linear.

   - **Filter**

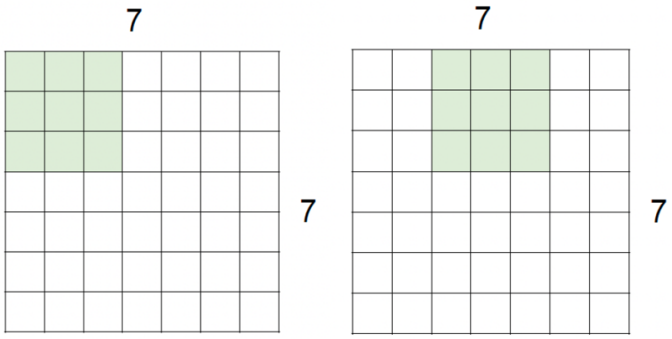


   - A filter is applied to the image and outputs a dot product of the two.



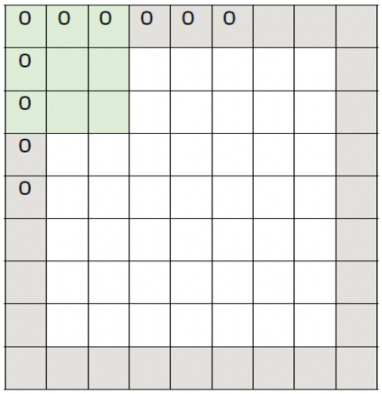
- Depending on how many filters we have, the output will have different number of activation maps (for example, 6 filters → 6 activation maps)

- **Stride**



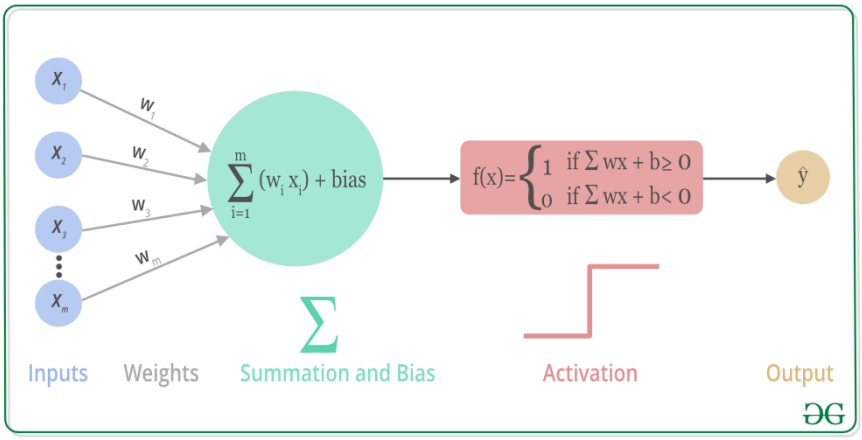
- Stride decides how many pixels you jump after applying the filter
- For example, in the picture above, the stride is 2.

- **Zero-padding**



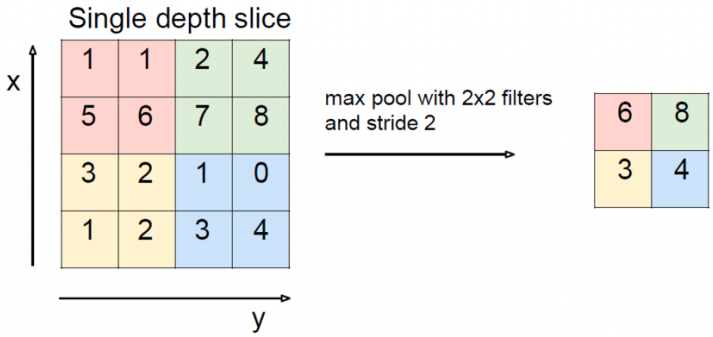
- Zero padding is a technique where we have 0 values outside the boundaries of the image (matrix) in order to:
    - Control the dimension from shrinking after applying a filter that's larger than 1x1
    - Avoid losing information at the boundaries

- **Activation function**



- takes the output from the previous cell, applies the function, and then creates and output that could be taken as an input for the next cell (or it may be the final output)
- Some of the activation functions include: ReLU, Sigmoid, tanh, softmax, etc

2. **Pooling layer**
   - Reduces the dimension of the feature/activation map in order to decrease the computing time



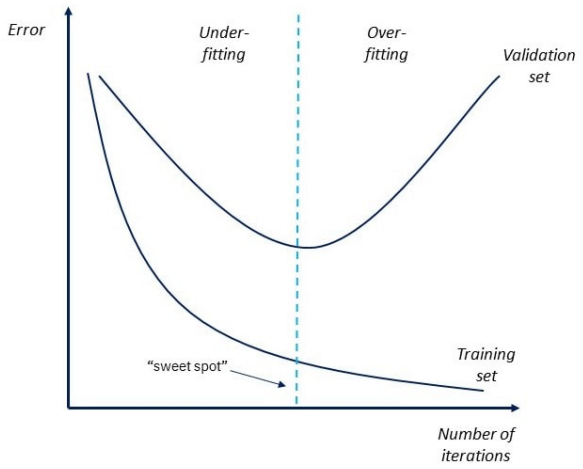
   - In the image above, we are using max pooling (retaining the maximum value within the filtered area)
   - Pooling is done on each feature map independently
3. **Dense layer**
   - Classifies the feature maps created/reduced by convolutional layer/pooling layer.
   - Activation function (explained in convolutional layer section)

We also specify which **loss functions** and **optimizers** we want to use for the model. The loss function calculates the error between the true and the predicted values (classes). Depending on which loss function we decide to use, we will have different error values. The optimizers are used to update the weights and biases of the model.

There are also **epochs** and **batch sizes** in the model. Epoch specifies how many times you want to train the model on the given dataset. Batch size is the size you want the training data to be

grouped in for the input. For example, if the batch size is 32, the first 32 images will be in one batch, and the next 32 images will be in another batch and so on.

One of the problems we may face in CNN is **overfitting**. It may overfit the train data that the accuracy on train data will be high but the accuracy on the test dataset may become lower. (Please refer to the figure below)



Some of the methods that could be used to reduce overfitting are:

- Dropout (randomly get rid of the nodes with its previous information)
- L2 = weight decay (penalizing big weights)
- Early-stopping (stop the fitting process if there's no improvement after some time)

Overall CNN process (summary)
We start out with the train dataset. It goes through multiple layers such as the convolutional layers and pooling layers. You can specify parameters/hyperparameters (number of filters, kernel sizes, loss functions, optimizer, etc.) for the layers. The layers will capture different patterns (weights and biases) at different levels. At the end of the model, the features will be classified into the number of groups you specify for the last dense layer.

➢ Detailed explanation of your code logic.

1. Mount Google Drive (computing done on Google Colab)
2. Load the train/test data using sio.loadmat function
3. Take a look at the data

```
# check what the loaded data looks like
print(type(train_data))
print(train_data.keys())
```

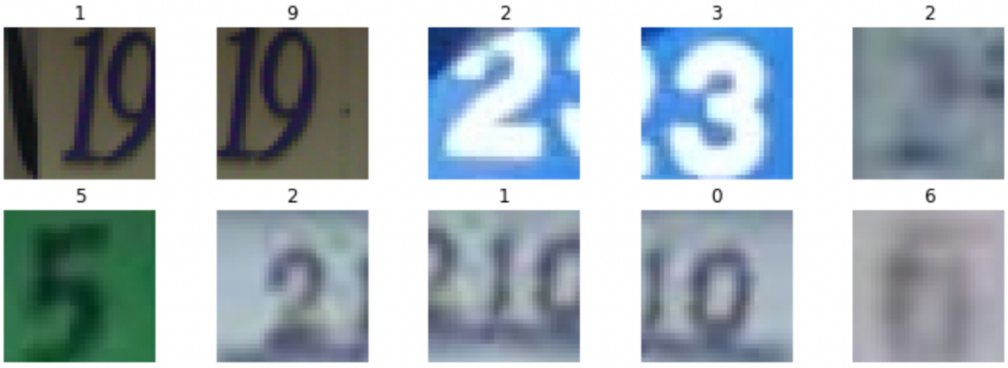
```
<class 'dict'>
dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
```

4. Separate the data into X_train, y_train, X_test, y_test
5. Scale the images to [0,1]

6. Transpose the data to have the number of images in the front
7. Check that the number of X_train/X_test corresponds to that of y_train/y_test
8. Check how many classes there are in the dataset

```
Shapes after tranposing the data:
X_train shape: (73257, 32, 32, 3)
train_label shape: (73257,)
X_test shape: (26032, 32, 32, 3)
test_label shape: (26032,)

Number of classes: 10
```

9. Convert 10 to 0 (for the next step)
10. Convert the class vectors to binary class matrices
11. Display the images with corresponding labels to take a look at the data we have
    a. First row is the first 5 train images
    b. Second row is the first 5 test images



12. Create a simple model
13. Fit the simple model and take a look at the evaluation
14. Plot the accuracy and loss of the model on train/test dataset
15. Create an improved model (load if there's model file)
16. Fit the model and take a look at the evaluation (save the model)
17. Plot the accuracy and loss of the improved model on train/test dataset
18. Make predictions on the test dataset to see the predictions
19. Visualize some of the test dataset to see the predicted/true labels
    a. Will display green if they match, red if they don't match

➢ Mention your starting model and how you choose that
➢ Model evaluation
➢ Explain how you reached final model from starting model with intermediate stages as well (including visualization)

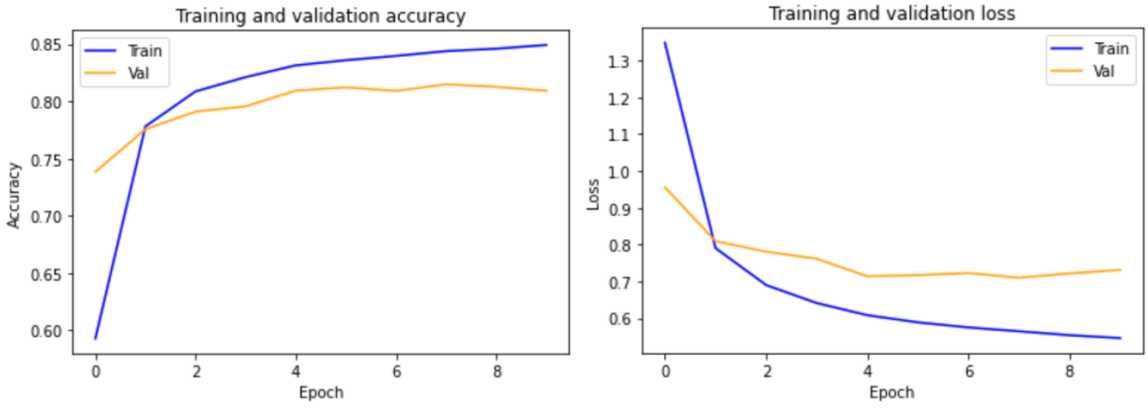
In this section, I will write about the models (starting and improved), evaluation (test accuracy), and how I reached the model with some visualizations

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 16)        448

 max_pooling2d (MaxPooling2D  (None, 15, 15, 16)       0
 )

 flatten (Flatten)           (None, 3600)              0

 dense (Dense)               (None, 10)                36010

=================================================================
Total params: 36,458
Trainable params: 36,458
Non-trainable params: 0
_________________________________________________________________
```

This is the model summary of my starting model. I wanted to start out with a simple model so that the improved model has a clear difference from the improved model. It contains one convolutional layer (with no padding, which results in output shape of 30x30x16 instead of 32x32x16), pooling layer, flattening layer (for converting the data into one dimension), and dense layer.

```
814/814 [==============================] - 3s 4ms/step - loss: 0.7313 - accuracy: 0.8093
[0.7312996983528137, 0.8092731833457947]
```

The accuracy on the test dataset was 80.9%.

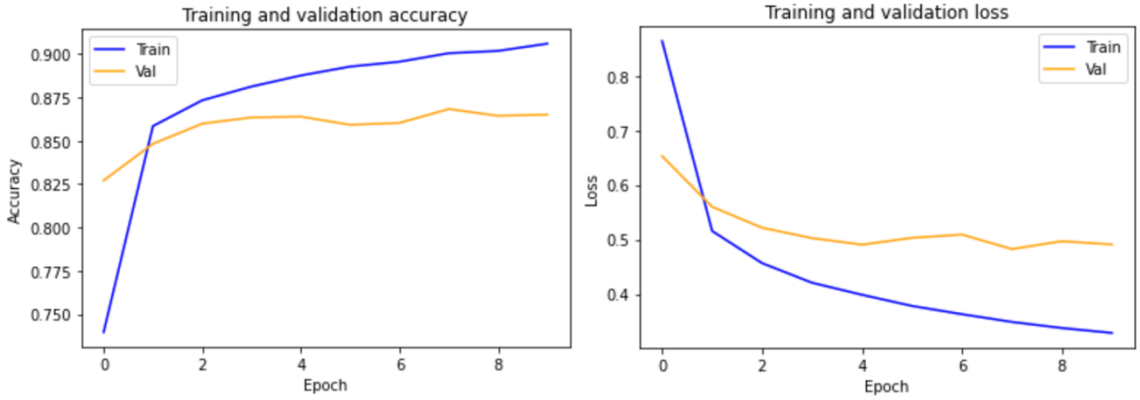


From the plots above, we can see that even though the accuracy and loss continue to improve for training dataset, the accuracy and loss plateaus at a certain level for the test dataset. The accuracy for the test dataset plateaus at around 81%.

For the intermediate models, there are several factors that I played around with. These factors include layers and parameter values.

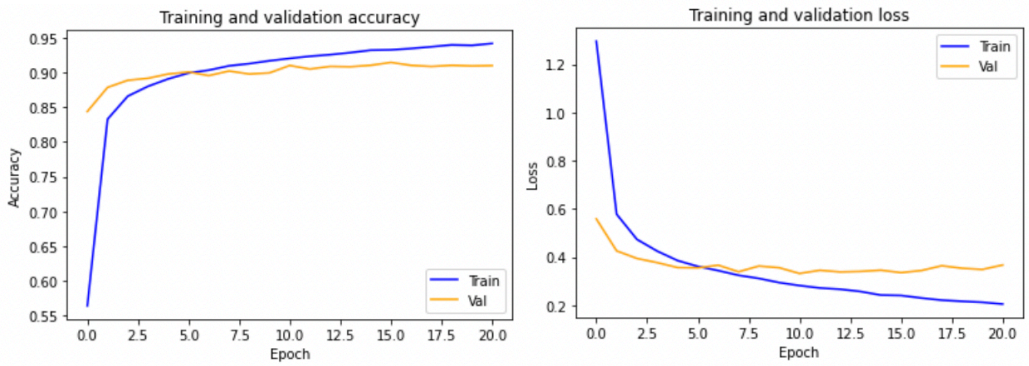
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 3136)              0

 dense_1 (Dense)             (None, 10)                31370

=================================================================
Total params: 50,762
Trainable params: 50,762
Non-trainable params: 0
```

This is a model summary of one of the intermediate models that I tested. I used different number of filters with another convolutional layer and another pooling layer added. The plot below is the accuracy and loss plot for the intermediate model. We can see that the test accuracy for the intermediate model plateaus at around 86%.




```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_6 (Conv2D)           (None, 3, 3, 256)         409856

 max_pooling2d_6 (MaxPooling  (None, 1, 1, 256)        0
 2D)

 dropout_3 (Dropout)         (None, 1, 1, 256)         0

 flatten_2 (Flatten)         (None, 256)               0

 dense_4 (Dense)             (None, 256)               65792

 dropout_4 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 84)                21588

 dropout_5 (Dropout)         (None, 84)                0

 dense_6 (Dense)             (None, 10)                850

=================================================================
Total params: 517,478
Trainable params: 517,478
Non-trainable params: 0
_________________________________________________________________
```

This is the model summary of the final model. I've added more convolutional layers and pooling layers with different number of filters, kernel sizes, padding options, and strides. For this model, I have also added dropout layers to refrain from overfitting. I added two more dense layers to detect different levels of the output features. Another factor I added that cannot be seen from the model summary is the early stopping method (also in order to reduce overfitting)



Above is the accuracy and loss plot of the final model. We can see that the accuracy of the model on the test dataset plateaus around 91%, which is what we see in the test dataset evaluation. When we run the evaluate function on the test dataset with the model we trained, we get the accuracy of 91.5%.

```
814/814 [==============================] - 5s 5ms/step - loss: 0.3398 - accuracy: 0.9149
[0.3398139774799347, 0.9149124026298523]
```

* All models have a default batch size of 32.
* Details about the number of filters, kernel sizes, padding options, strides, optimizers, etc are in the code.

➢ What are the difficulties faced while implementing the model?

One of the difficulties I faced while implementing the model include the time it takes to train the model. For a simple model, it took less computing time, but as the model got more complex, it meant that there was more computing that needs to be done due to increased number of layers and filter numbers.

Another problem I encountered was that choosing the layers, number of filters, kernel sizes, padding options, strides, etc were all done by trial and error, so it was difficult to find the parameters/hyperparameters that work better. I had to plug and chug the numbers and train the model to see if it performed better or worse, and due to the first problem I mentioned in the previous paragraph, it took a long time to find out the right model.

➢ How can you improve the model further?

There are several ways I could do to improve the model

1. If I had better computing power, I could try out more models with more layers and parameters/hyperparameters.
2. I could use data augmentation method to increase the number of training dataset. Data augmentation performs a transformation on the training dataset I already have, which I can use as additional training data.
3. I could also try transfer learning method. Transfer learning is a method where I use a model that already has knowledge in solving one problem and applying it to a different problem. This method is also faster, so it could help out with the first problem I listed above.