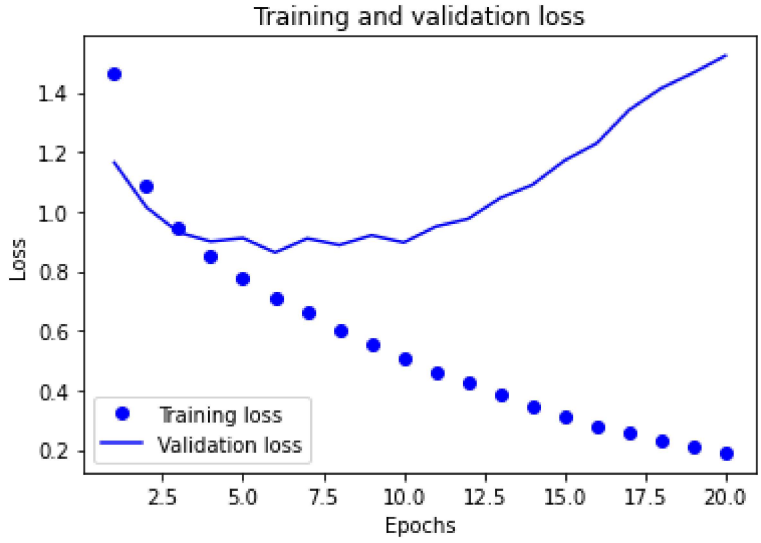
```
val_loss_values = history_dict['val_loss']

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7feac82bc908>
```




# 1. Data

I used the CIFAR-10 dataset: https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz. 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 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

I chose this dataset because it contains colorful images of common objects.

### Reference

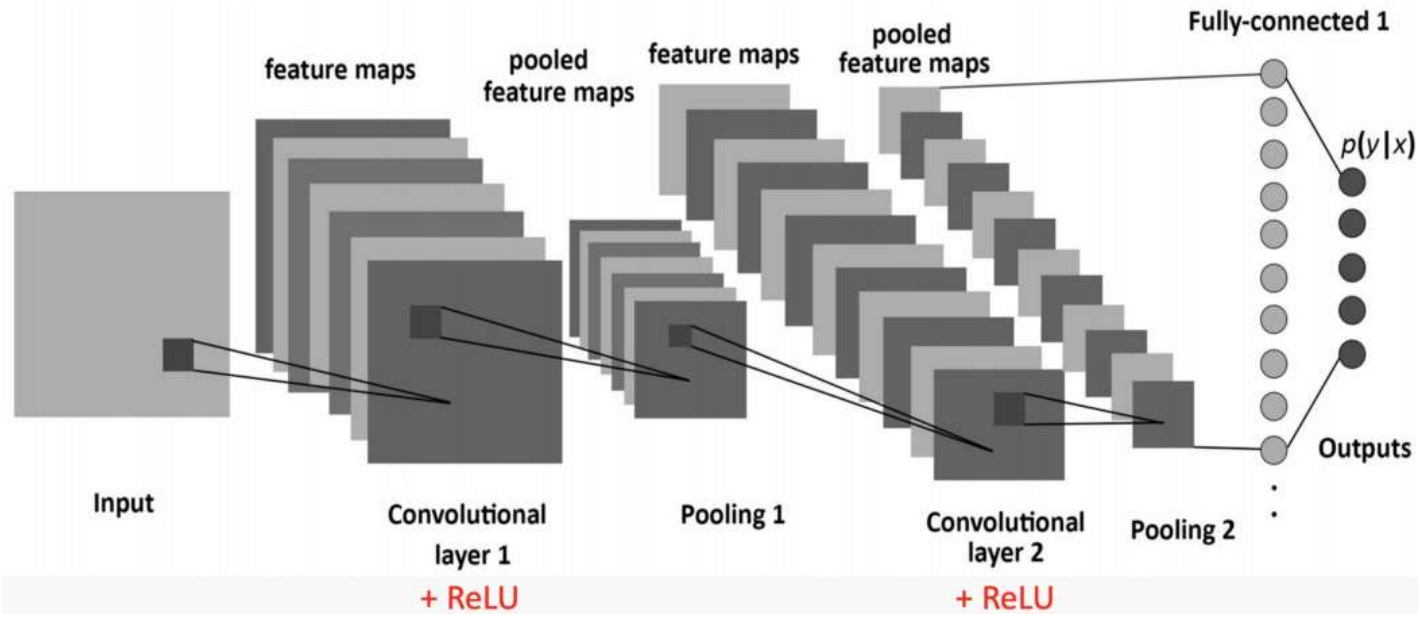
- Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

## 2. Network

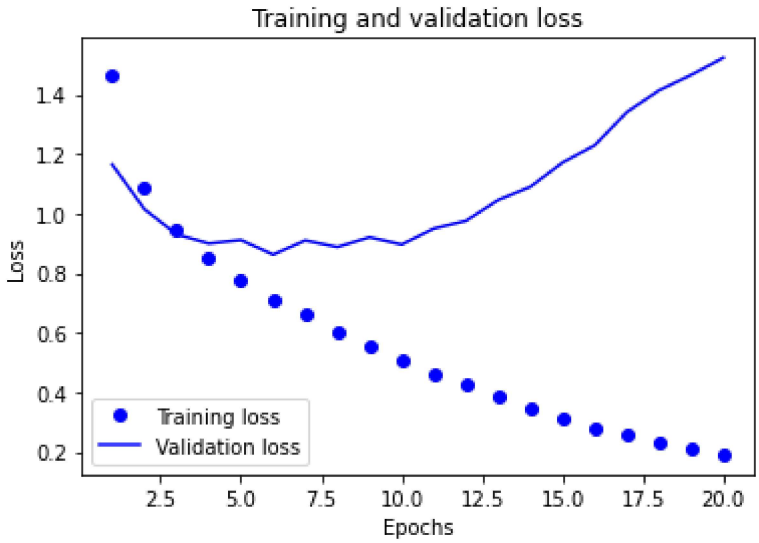
2 convolution and max pooling layers and 2 dense layers.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_8 (Dense)              (None, 100)               460900
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1010
=================================================================
Total params: 537,558
Trainable params: 537,558
Non-trainable params: 0
_________________________________________________________________
None
```

The CNN network is similar to the one below with 1 more dense layer.



# 3. Training




Batch size is 50. 20 Epochs of training.

# 4. Validation

Training accuracy goes to around 93% in 20 epochs. And testing accuracy goes to 70% in around 6 epochs and stays there. A slight indication of overfitting as the training accuracy surpasses 90% and the testing accuracy dips slightly below 70%.

# 5. Discussion

I used a basic CNN network with 2 convolutional and max pooling layers and 2 dense layers. The 1st convolutional and max pooling layer has 64 kernels, and the 2nd 128.

I have also tried a narrower network of similar depth, with 32 and 64 kernels in the convolutional and max pooling layers and 1 dense layer. We get lower training accuracy around 78% in 15 epochs but similar testing accuracy of 70%.

I found that widening the network doesn't change the testing accuracy but increases the training accuracy.