Homework 3

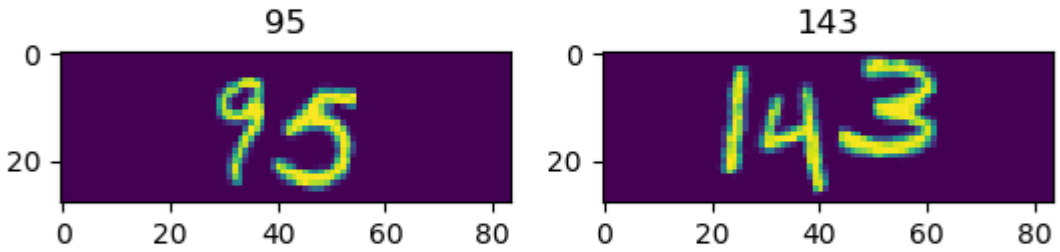
The homework had 2 tasks:

- Image classification
- Addition

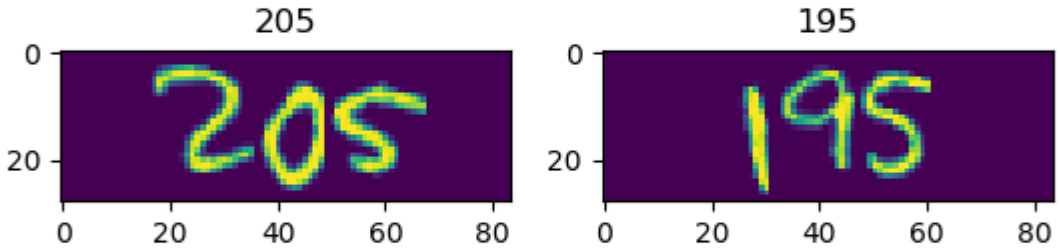
Dataset

The dataset consists of images of numbers formed using digits from the MNIST dataset. The maximum number in the dataset is 255.

Training Dataset



Test Dataset



Classification

For the classification task I tried four convolutional neural network arhitectures inspired by state of the art arhitectures.

Hyperparameters

For all the following networks I used the following hyperparameters:

Batch size: 512 It first I tried some smaller batch sizes but there was not enough data for the network. Steps per epoch: 500 Also chosed a bigger steps per epoch so the network would have more data.

Optimizer: Adam Adam is a good optimizer for getting good results fast. I also used the default parameters for Adam.

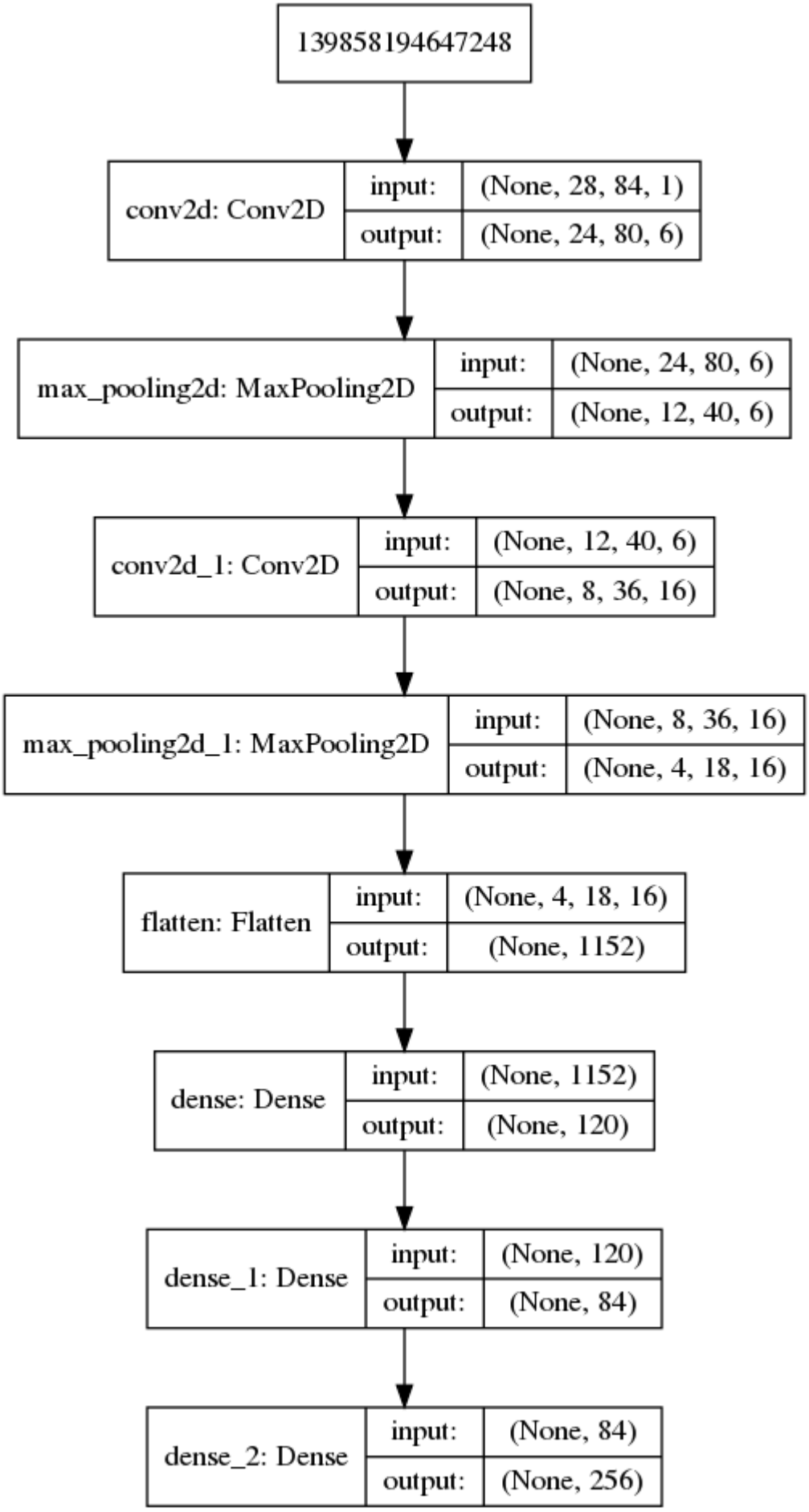
Epochs: 10 I trained all the networks for 10 epochs. In the first epochs the networks had a big boost in accuracy, then it slowly incresed.

Activations On convolutional and dense layers, except the last one I used ReLU activation function. On the last layer I used softmax because it's a classification problem.

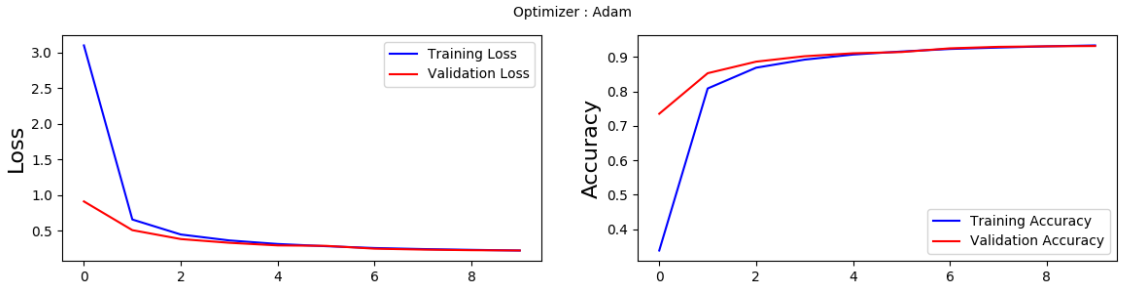
Lenet

The LeNet architecture was first introduced by LeCun et al. in their 1998 paper, Gradient-Based Learning Applied to Document Recognition. As the name of the paper suggests, the authors' implementation of LeNet was used primarily for OCR and character recognition in documents.

Model Arhitecture



Training history



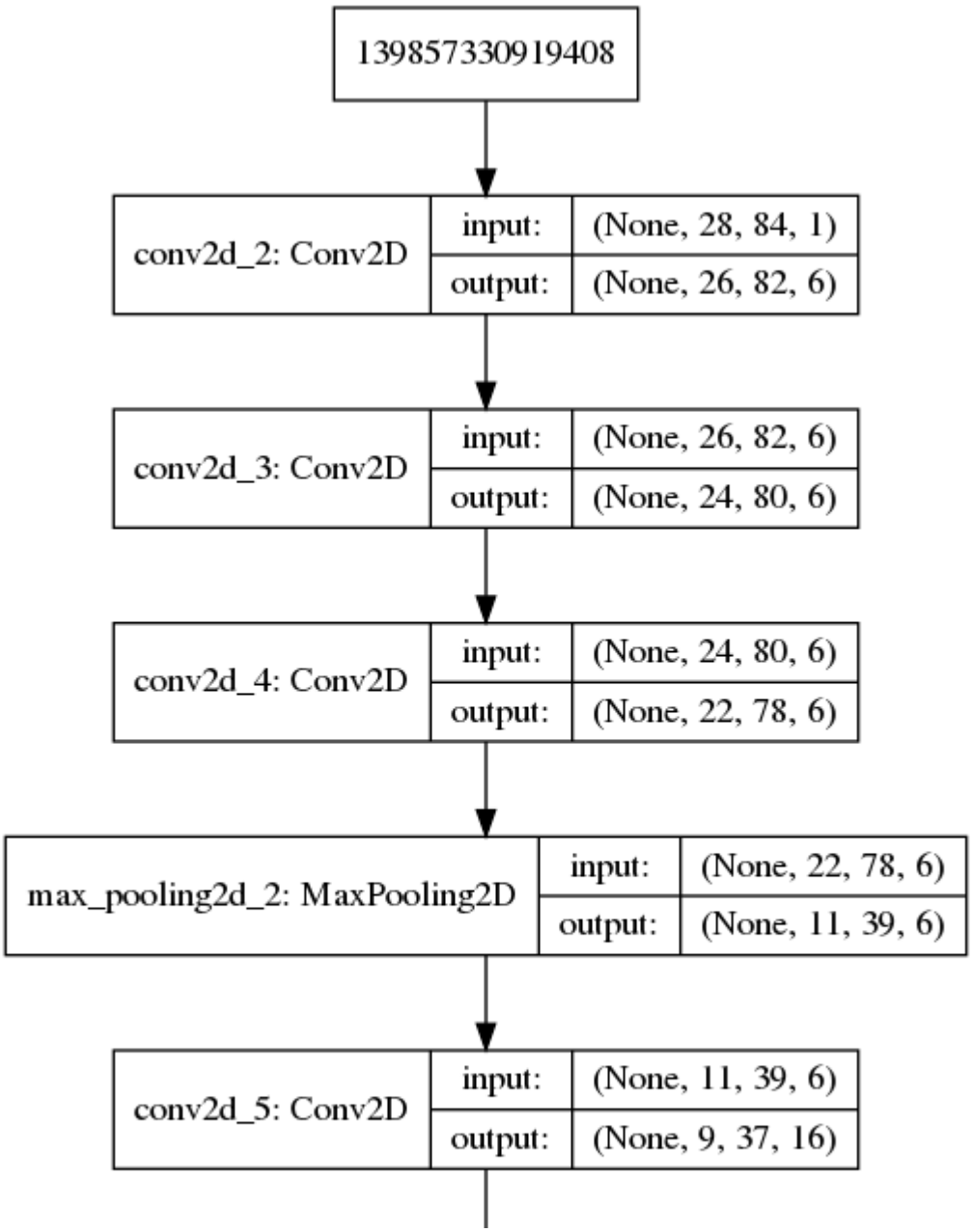
Results

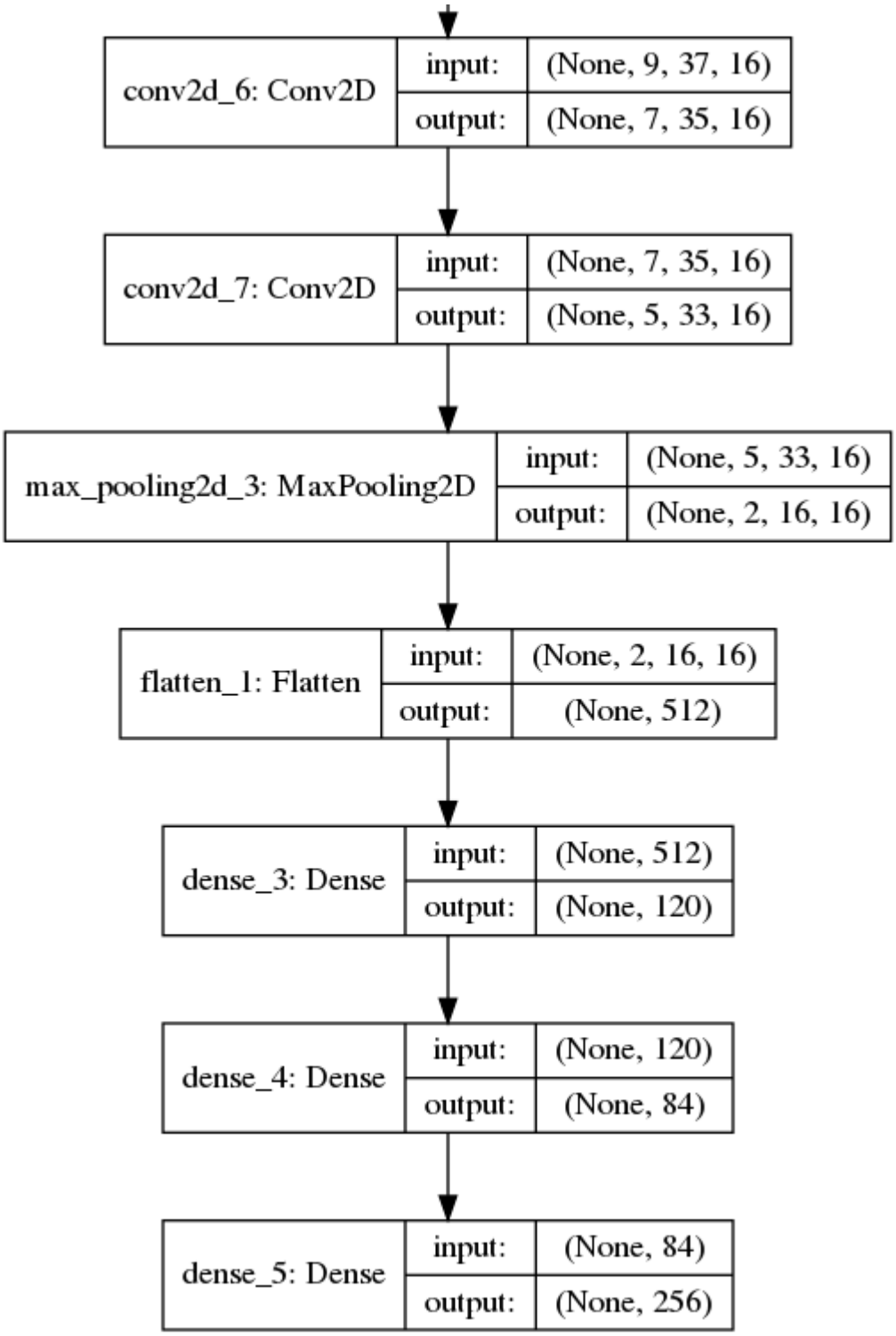
The network achived a test accuracy of 93.84%.

Lenet with a vgg flavour

Originally the lenet arhitecture had a big kernel size on the convolution layers so I multiplied the layer with smaller kernel size.

Model Arhitecture





Training history

Results

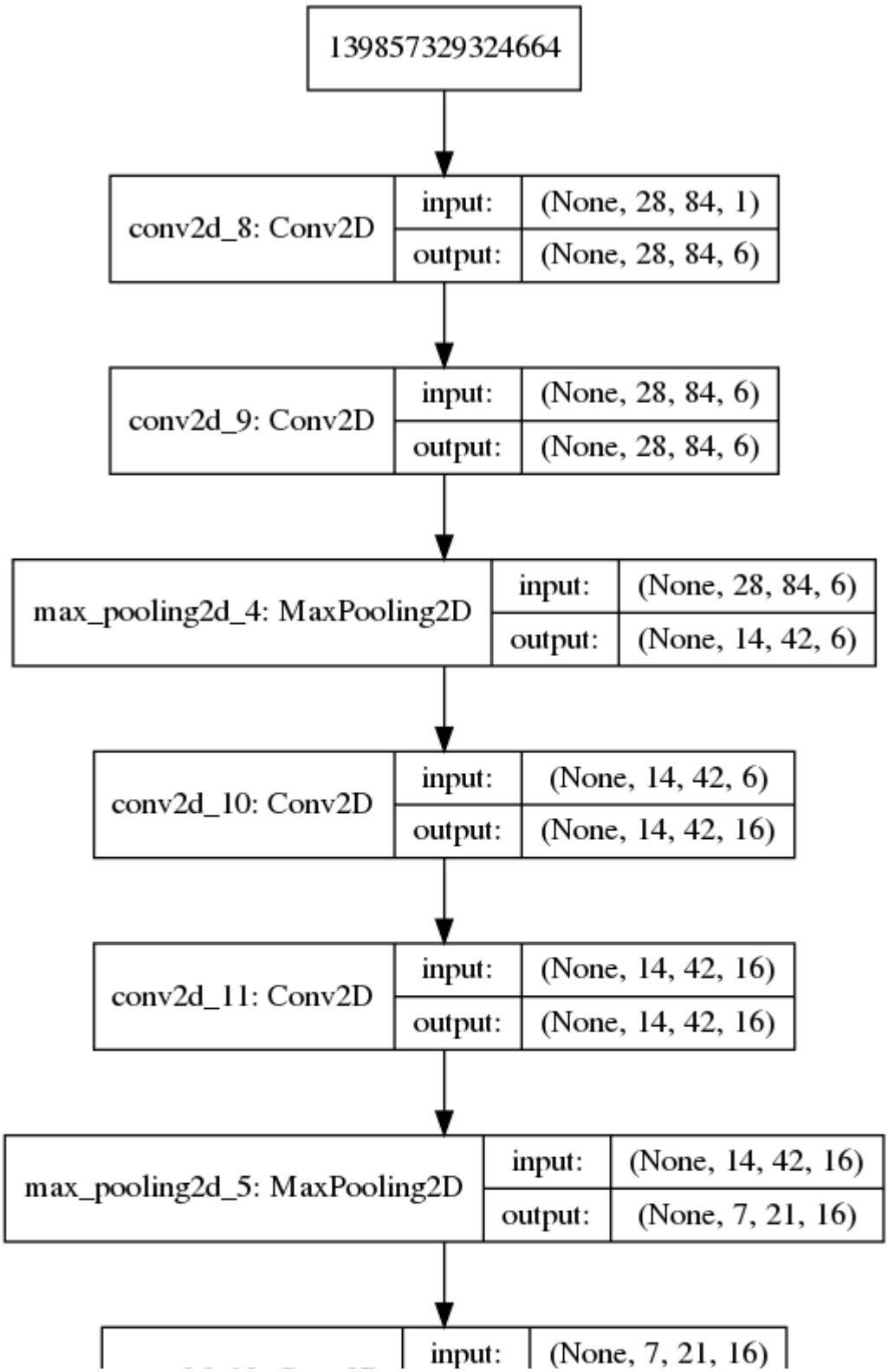
The network achived a test accuracy of 96.15%.

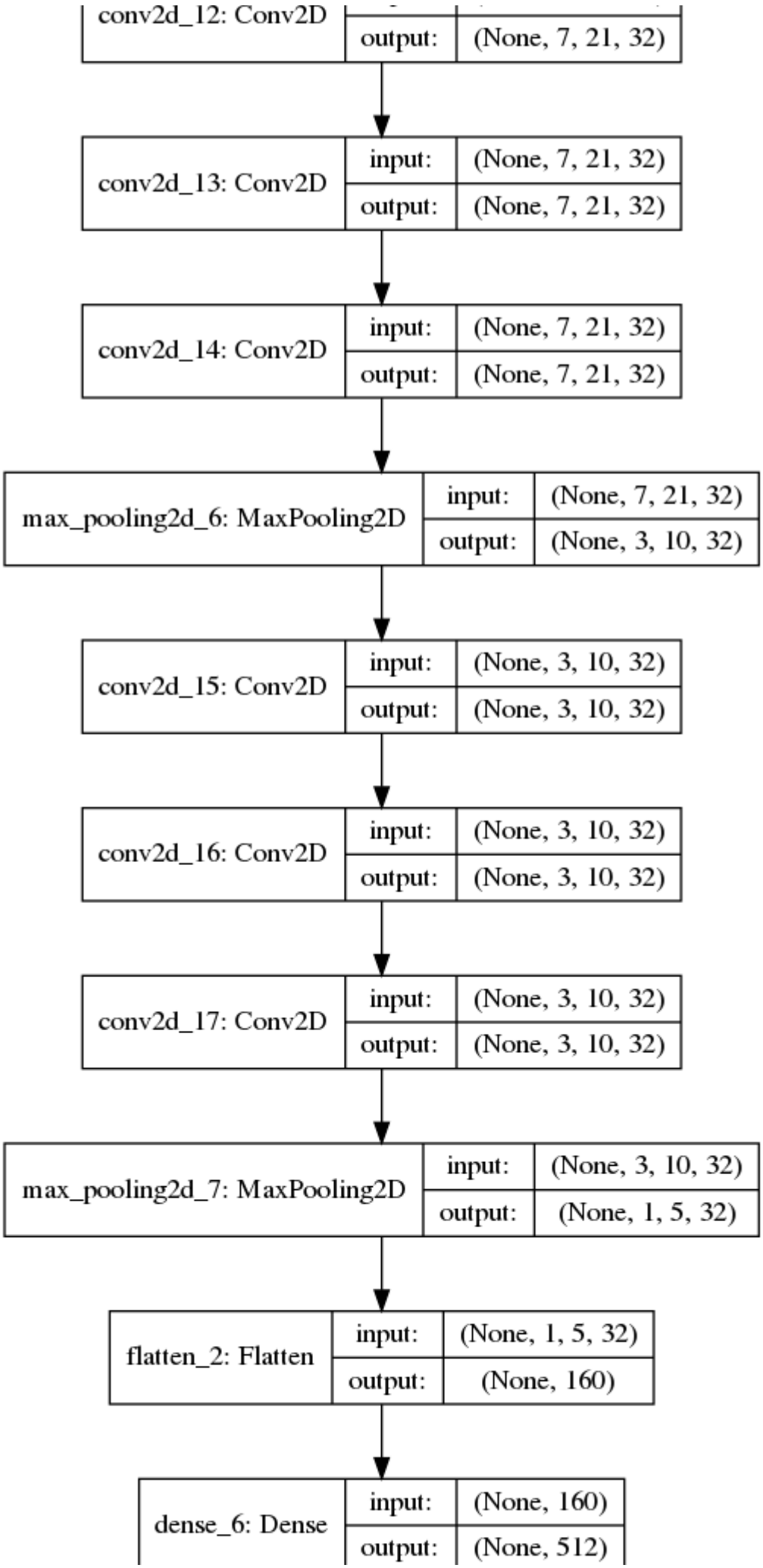
VGG style networks

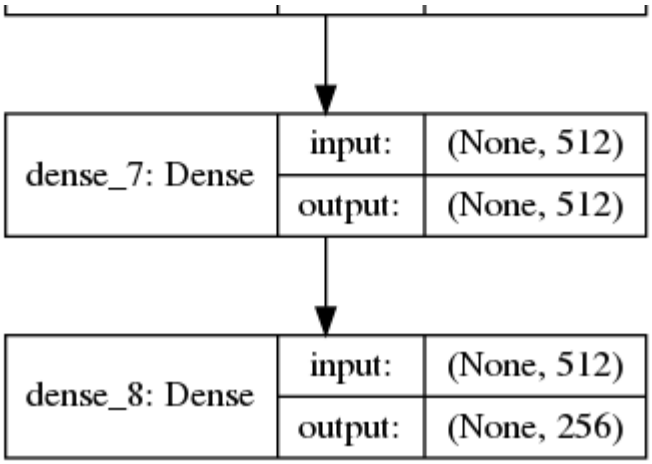
I tried two networks close to the original VGG arhitecture. The networks have four convolutional blocks followed by two Dense layers with 512 units and the classification layer.

First network have filter sizes of 6, 16, 32 and 32. The second network have filter sizes of 32, 64, 128 and 128.

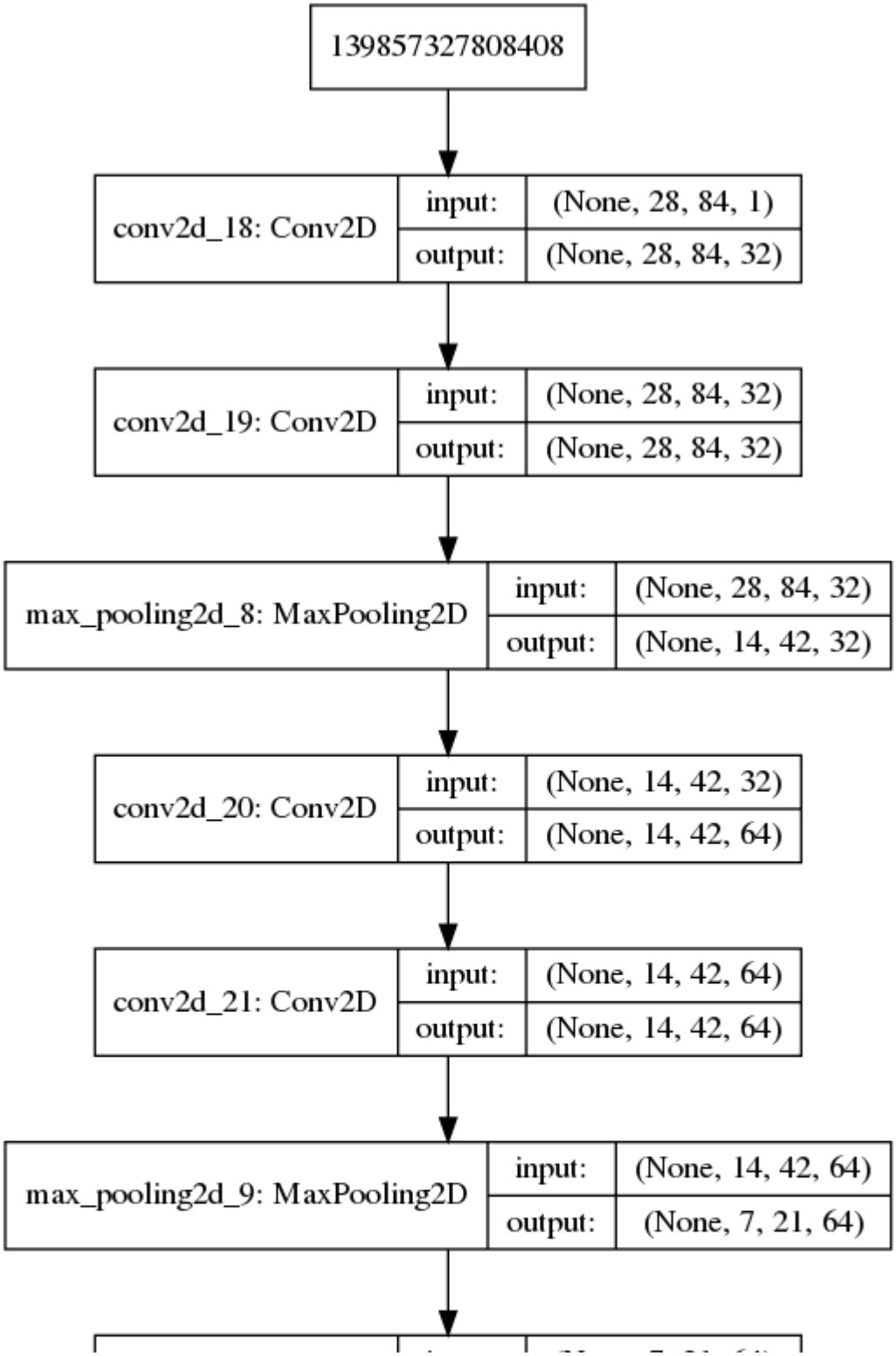
Model Arhitectures

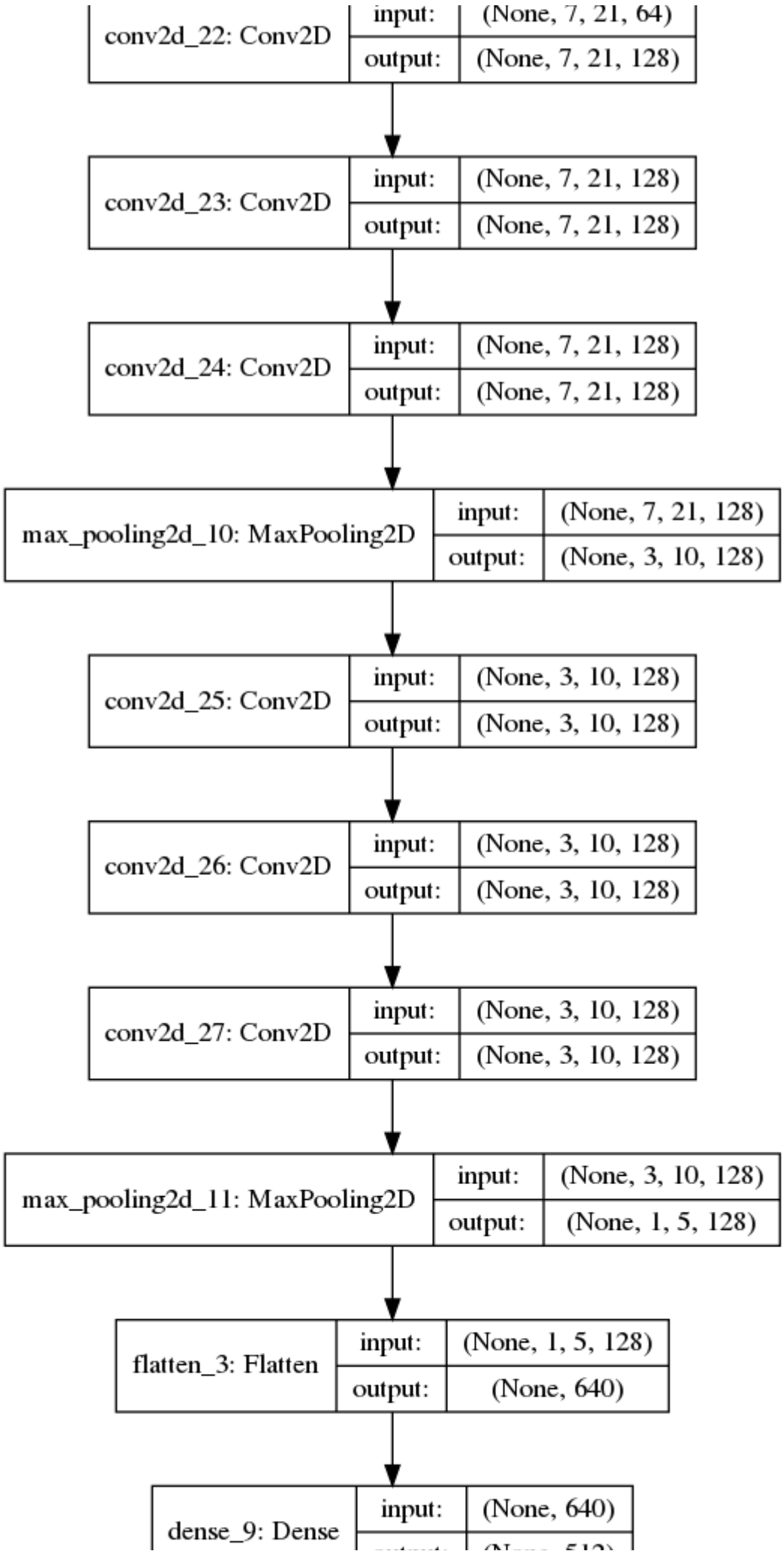


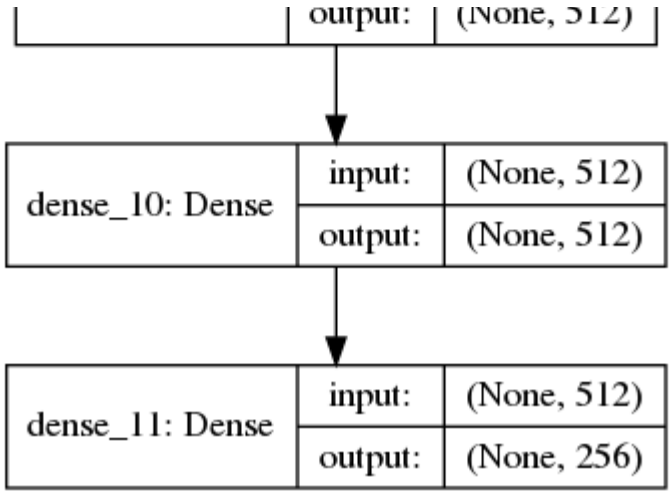

conv2d_12: Conv2D
output: (None, 7, 21, 32)
conv2d_13: Conv2D
input: (None, 7, 21, 32)
output: (None, 7, 21, 32)
conv2d_14: Conv2D
input: (None, 7, 21, 32)
output: (None, 7, 21, 32)
max_pooling2d_6: MaxPooling2D
input: (None, 7, 21, 32)
output: (None, 3, 10, 32)
conv2d_15: Conv2D
input: (None, 3, 10, 32)
output: (None, 3, 10, 32)
conv2d_16: Conv2D
input: (None, 3, 10, 32)
output: (None, 3, 10, 32)
conv2d_17: Conv2D
input: (None, 3, 10, 32)
output: (None, 3, 10, 32)
max_pooling2d_7: MaxPooling2D
input: (None, 3, 10, 32)
output: (None, 1, 5, 32)
flatten_2: Flatten
input: (None, 1, 5, 32)
output: (None, 160)
dense_6: Dense
input: (None, 160)
output: (None, 512)


dense_7: Dense
input: (None, 512)
output: (None, 512)
dense_8: Dense
input: (None, 512)
output: (None, 256)

First Model


139857327808408
conv2d_18: Conv2D
input: (None, 28, 84, 1)
output: (None, 28, 84, 32)
conv2d_19: Conv2D
input: (None, 28, 84, 32)
output: (None, 28, 84, 32)
max_pooling2d_8: MaxPooling2D
input: (None, 28, 84, 32)
output: (None, 14, 42, 32)
conv2d_20: Conv2D
input: (None, 14, 42, 32)
output: (None, 14, 42, 64)
conv2d_21: Conv2D
input: (None, 14, 42, 64)
output: (None, 14, 42, 64)
max_pooling2d_9: MaxPooling2D
input: (None, 14, 42, 64)
output: (None, 7, 21, 64)

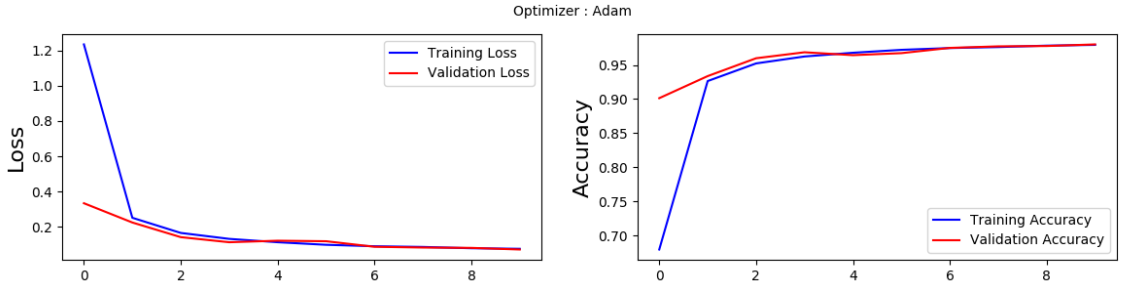

conv2d_22: Conv2D
input: (None, 7, 21, 64)
output: (None, 7, 21, 128)
conv2d_23: Conv2D
input: (None, 7, 21, 128)
output: (None, 7, 21, 128)
conv2d_24: Conv2D
input: (None, 7, 21, 128)
output: (None, 7, 21, 128)
max_pooling2d_10: MaxPooling2D
input: (None, 7, 21, 128)
output: (None, 3, 10, 128)
conv2d_25: Conv2D
input: (None, 3, 10, 128)
output: (None, 3, 10, 128)
conv2d_26: Conv2D
input: (None, 3, 10, 128)
output: (None, 3, 10, 128)
conv2d_27: Conv2D
input: (None, 3, 10, 128)
output: (None, 3, 10, 128)
max_pooling2d_11: MaxPooling2D
input: (None, 3, 10, 128)
output: (None, 1, 5, 128)
flatten_3: Flatten
input: (None, 1, 5, 128)
output: (None, 640)
dense_9: Dense
input: (None, 640)



Second model

Training history

First model



Second model

Results

First network has a test accuracy of 97.76% and the second of 98.34%.

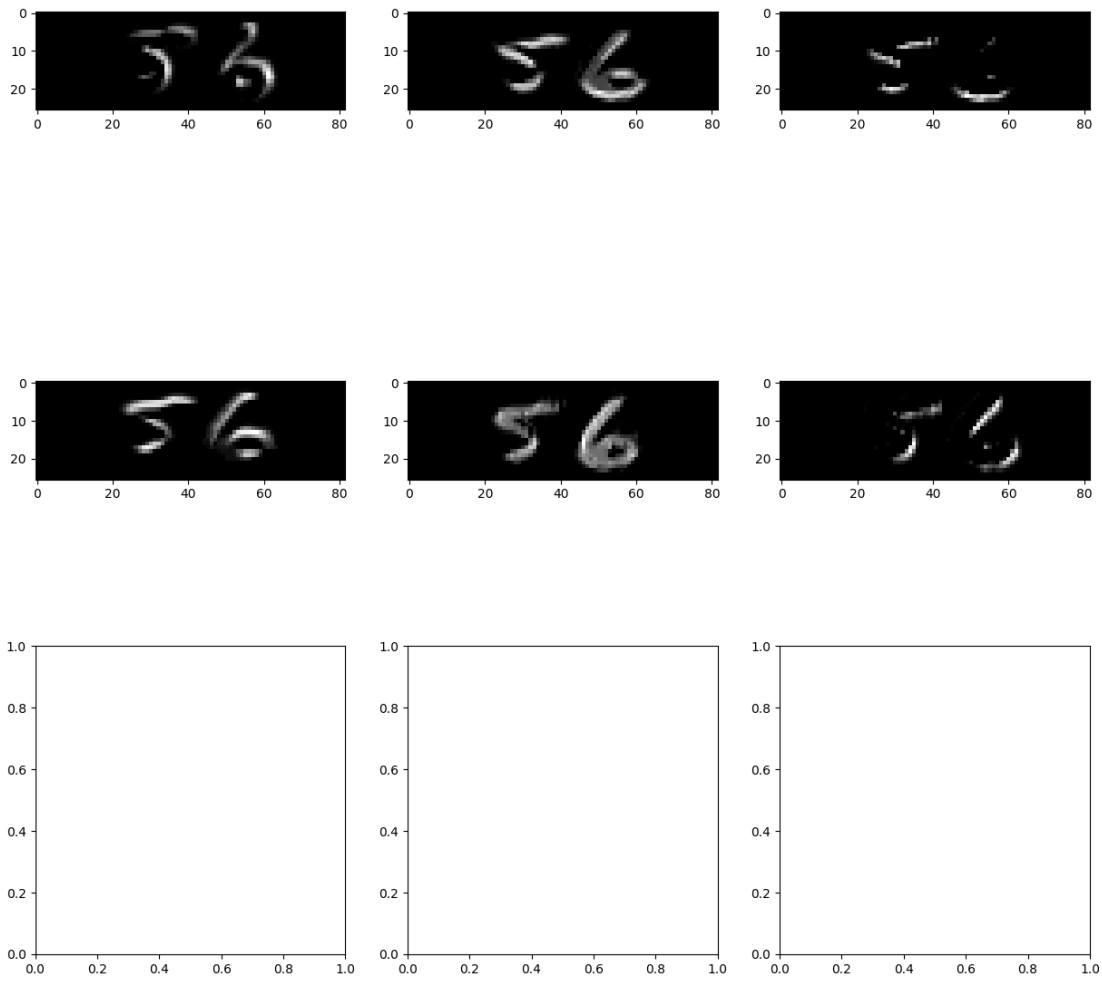
Results compared

Model Name	Learning Rate	Batch size	Epochs	Steps per epoch	Score	Accuracy
lenet	default	512	10	500	0.20424838069081305	0.9384375
changed_lenet	default	512	10	500	0.11745172668248414	0.9615625
vgg	default	512	10	500	0.08127159667015076	0.977671875
vgg_2	default	512	10	500	0.07867940024472773	0.9834375

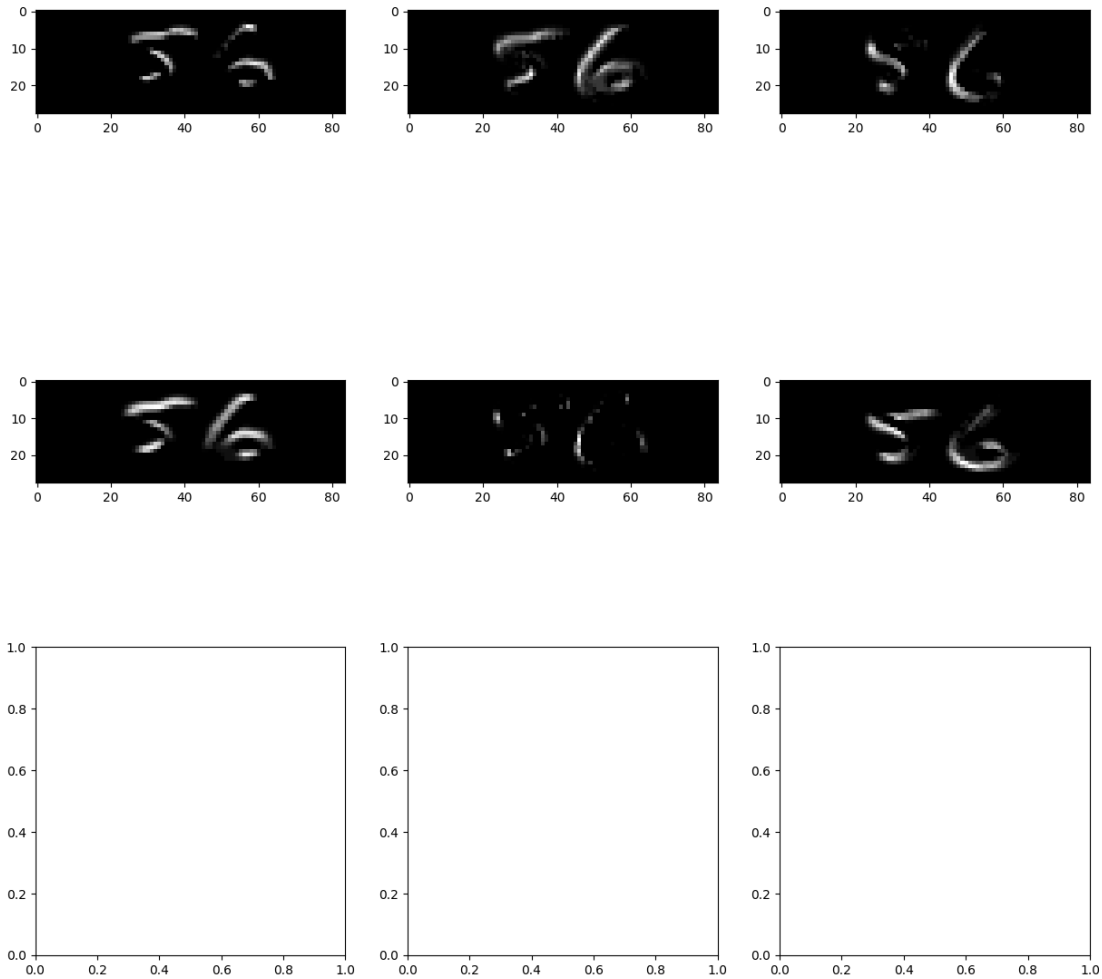
Activation filters compared

Here is a comparison of the first layer's activation for each model.

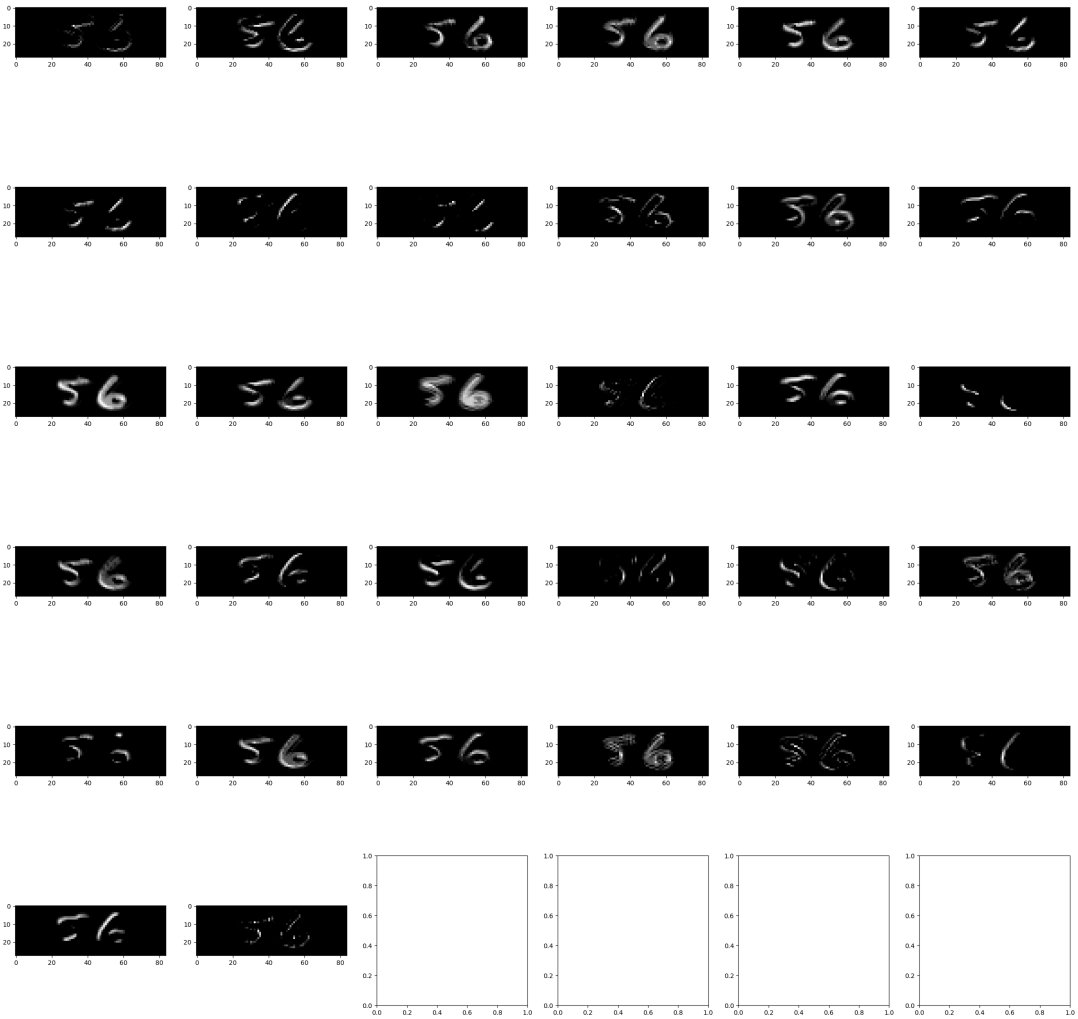
Changed Lenet



VGG



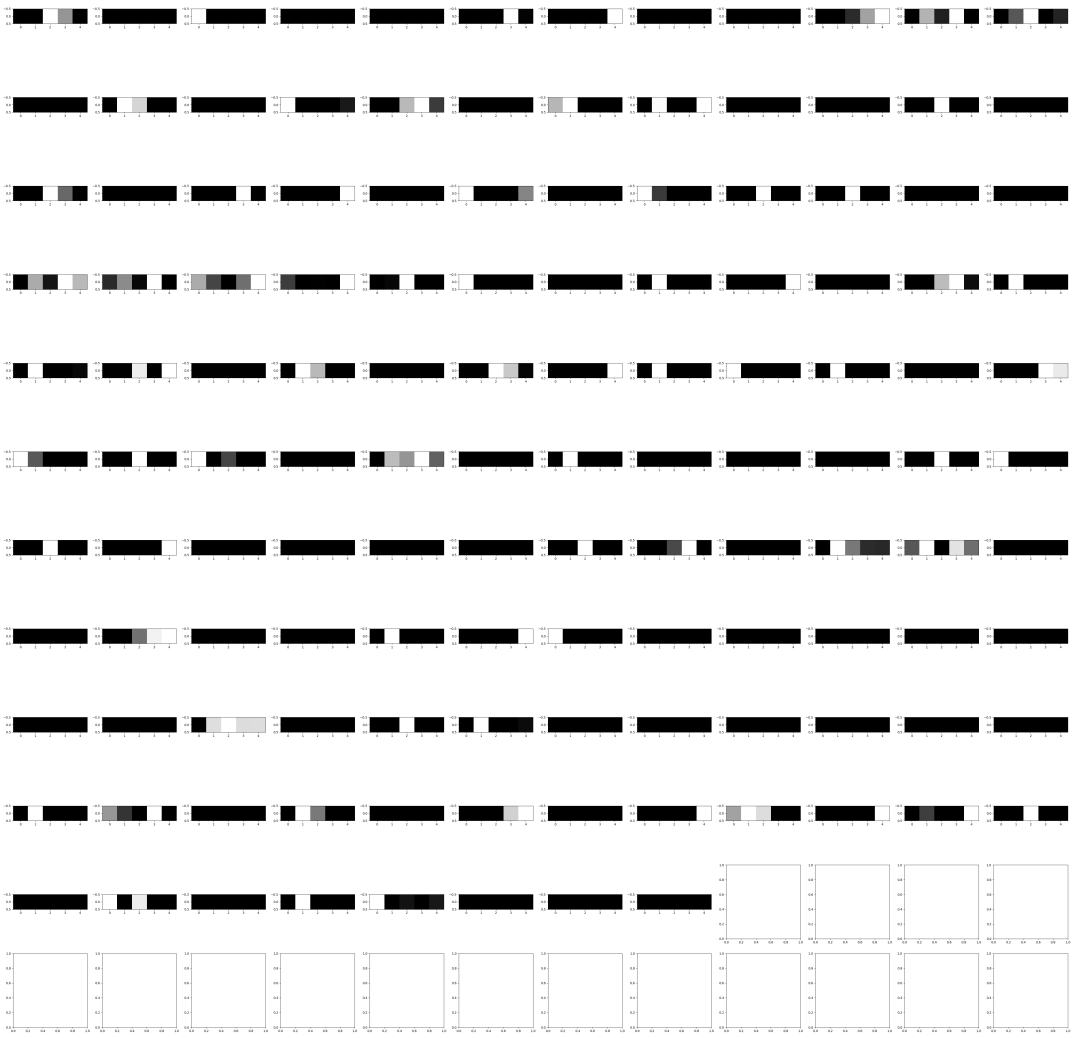
VGG2



And the last layer Changed Lenet

VGG

VGG2



Addition

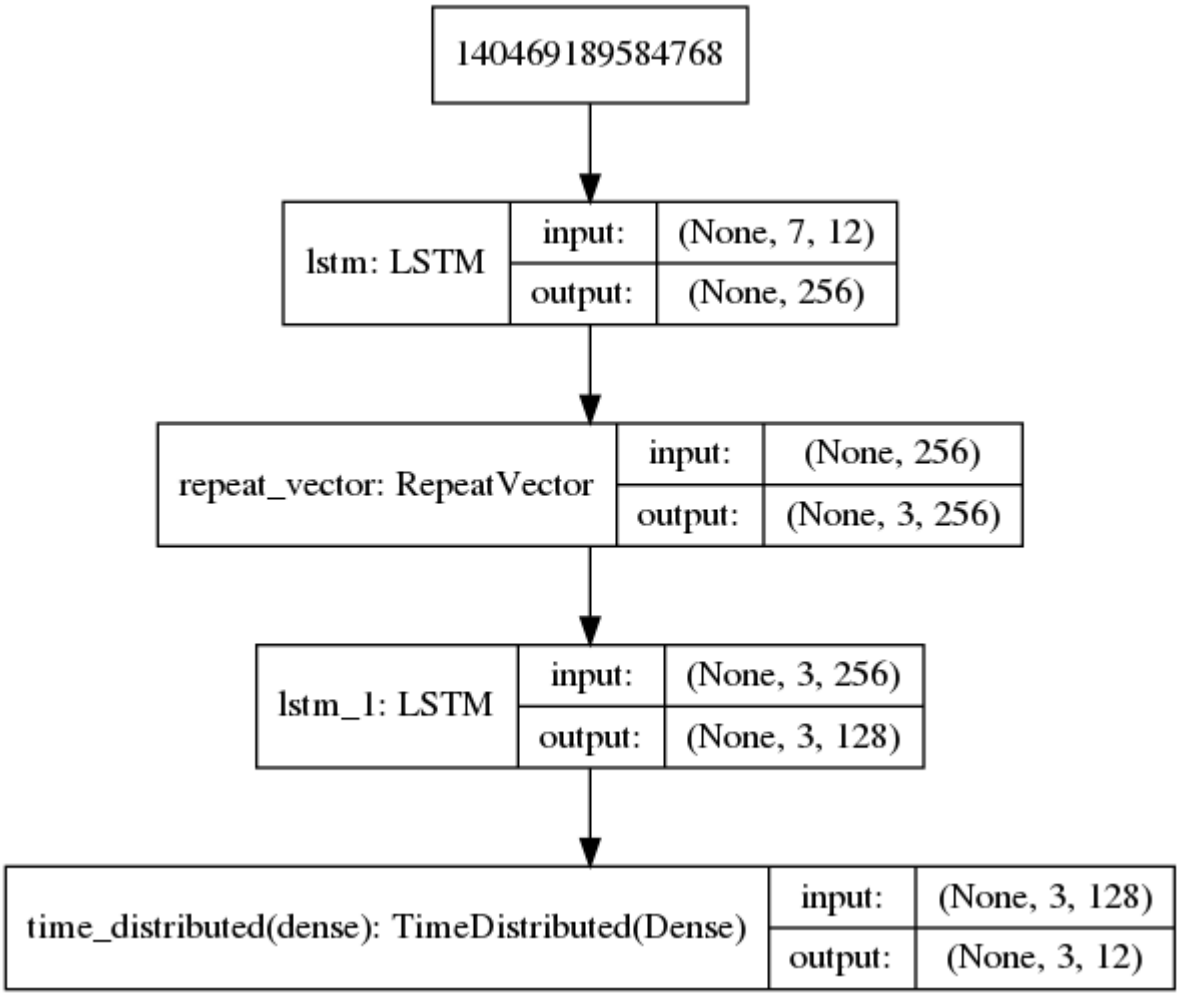
For the addition task I built a recurrent neural network using the encoder-decorer arhitecture.

Encoder Decoder arhitecture

We are solving this addition problem as a sequence to sequence problem so we need to transform the input data in a sequence. The original input consists of pairs of numbers(eg [10, 20]). First step is to transform the pair in a string ([10, 20] becomes ' 10+20'). The empty spaces before the string are used for the padding. The max size of the input is 7, 2 * 3 digits numbers and the '+' symbol. We need to work with numbers, so the next step is to hot encode the inputs. The alphabet consists of 12 characters so the input shape will be (7, 12), and the output shape will be (3, 12).

Model arhitecture

The model consists of 2 LSTM layers.



Results

Using this arhitecture, the test accuracy was 99.73%