M3 – Machine Learning for Computer Vision

Project: Deep learning classification - **Session 3**

Group 7: Guillem Capellera, Johnny Nuñez and Anna Oliveras

Tasks

Understanding the network topology

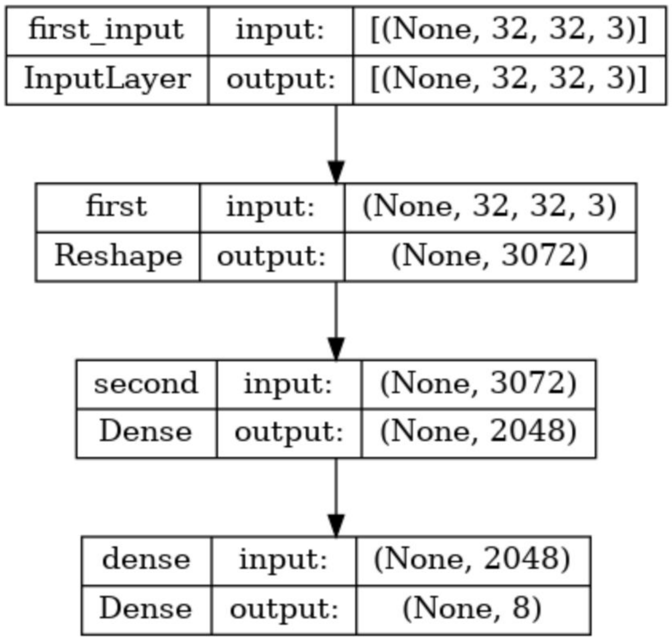
1. Add/change layers in the network topology
2. Given an image, get the output of a given layer
3. Manage to merge multiple outputs from a single image in an end to end network

Compare learnt features vs handcrafted features

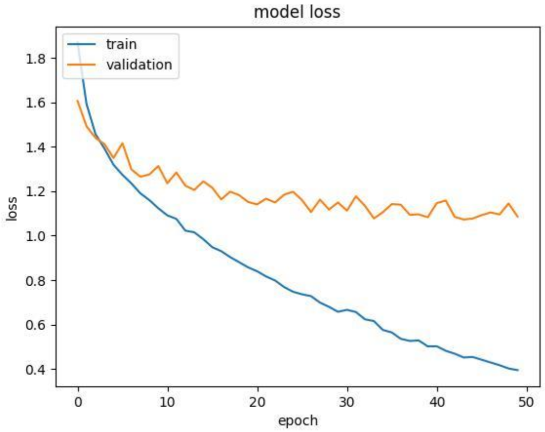
4. Extract a single feature from an input and apply to svm , compare to end to end network
5. Extract multiple features from an image and apply BoW , compare to end to end network

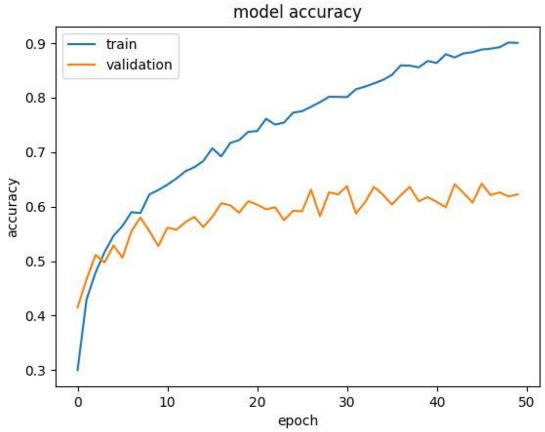
Experiments

- Code provided to us:
  - Data
    - Train: train folder
    - Validation: test folder
  - Architecture



Task 1: Initial network





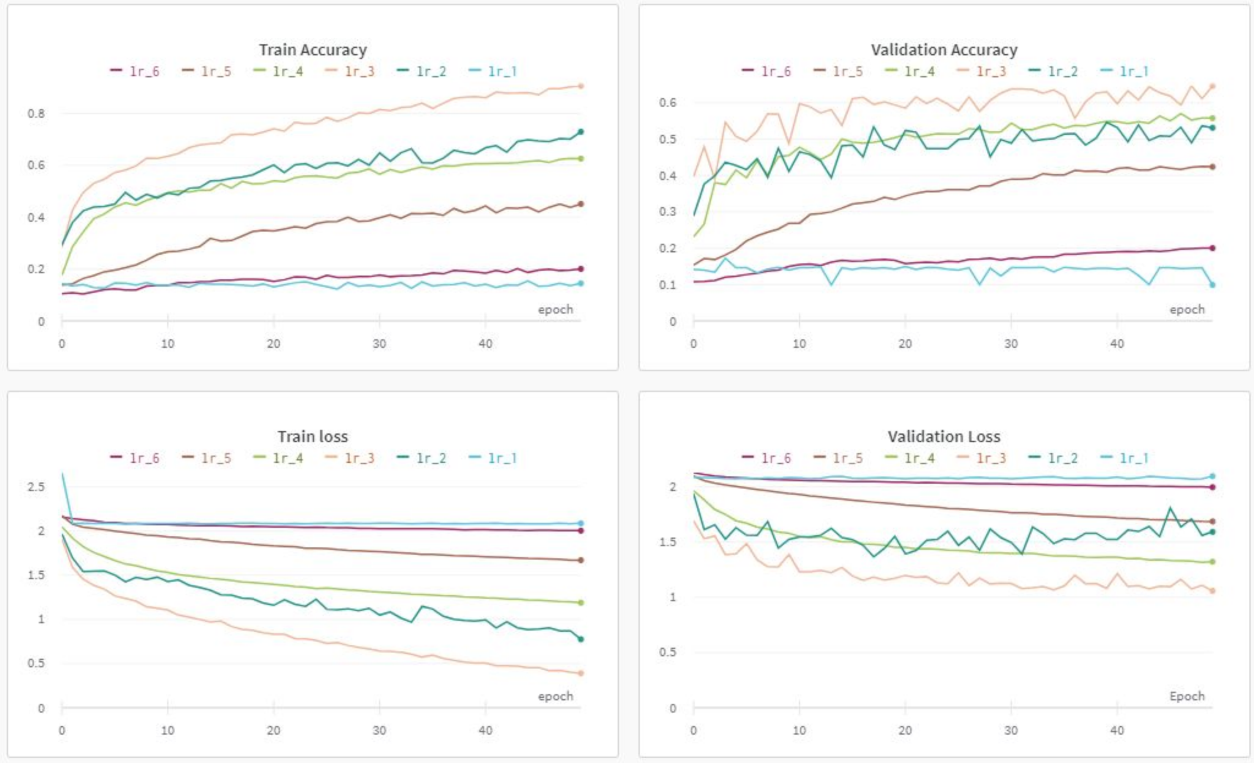

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
first (Reshape)              (None, 3072)              0

second (Dense)               (None, 2048)              6293504

dense (Dense)                (None, 8)                 16392

=================================================================
Total params: 6,309,896
Trainable params: 6,309,896
Non-trainable params: 0
```

After 50 epochs	**Train**	**Validation**
Accuracy	0.901	0.61
Loss	0.388	0.388

→ Big accuracy gap between Training and Validation

→ Validation accuracy does not improve with more epochs

Task 1: Changing the learning rate (LR)



Learning Rate

lr_1 = 0.1
lr_2 = 0.01
lr_3 = 0.001
lr_4 = 0.0001
lr_5 = 0.00001
lr_6 = 0.000001

The best result obtained is with the default LR (0.001), we obtain the best accuracy and lowest loss

Task 1: Changing the image size

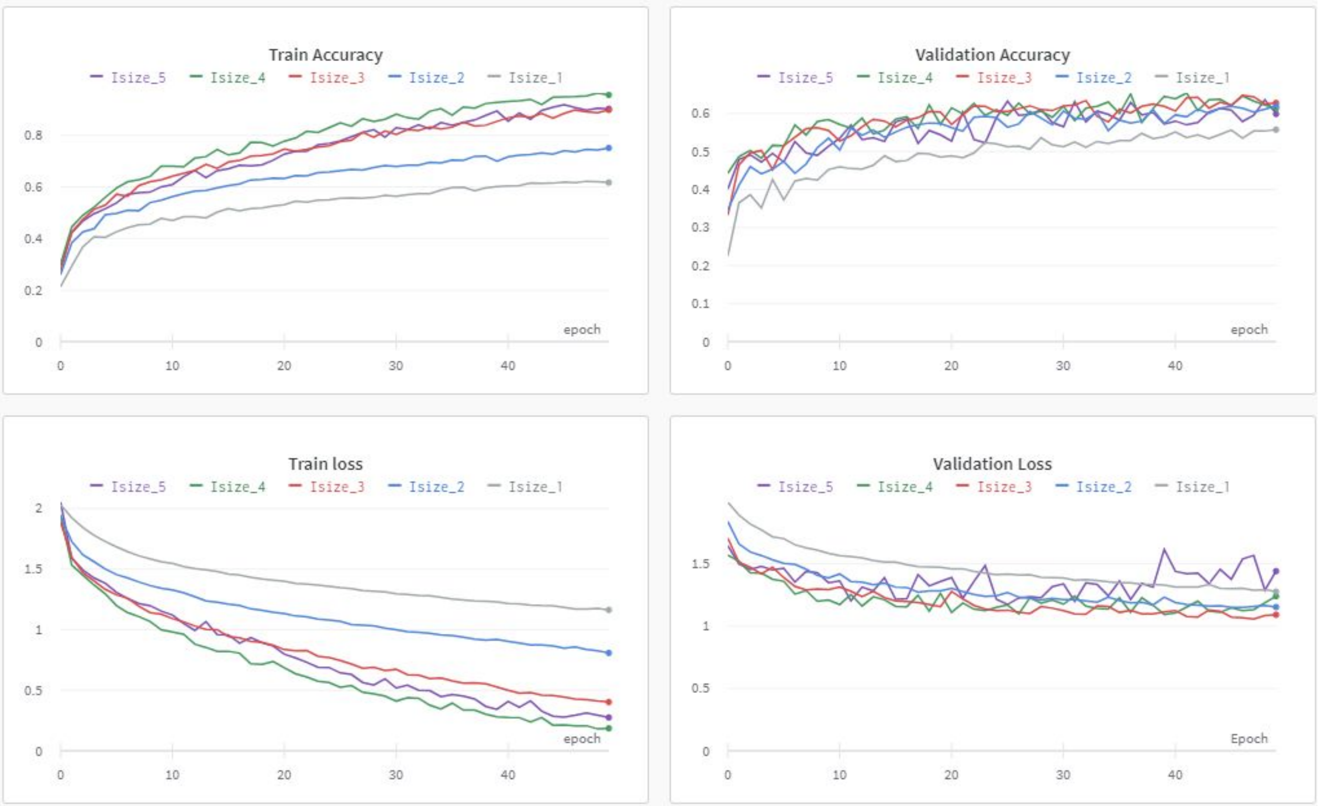
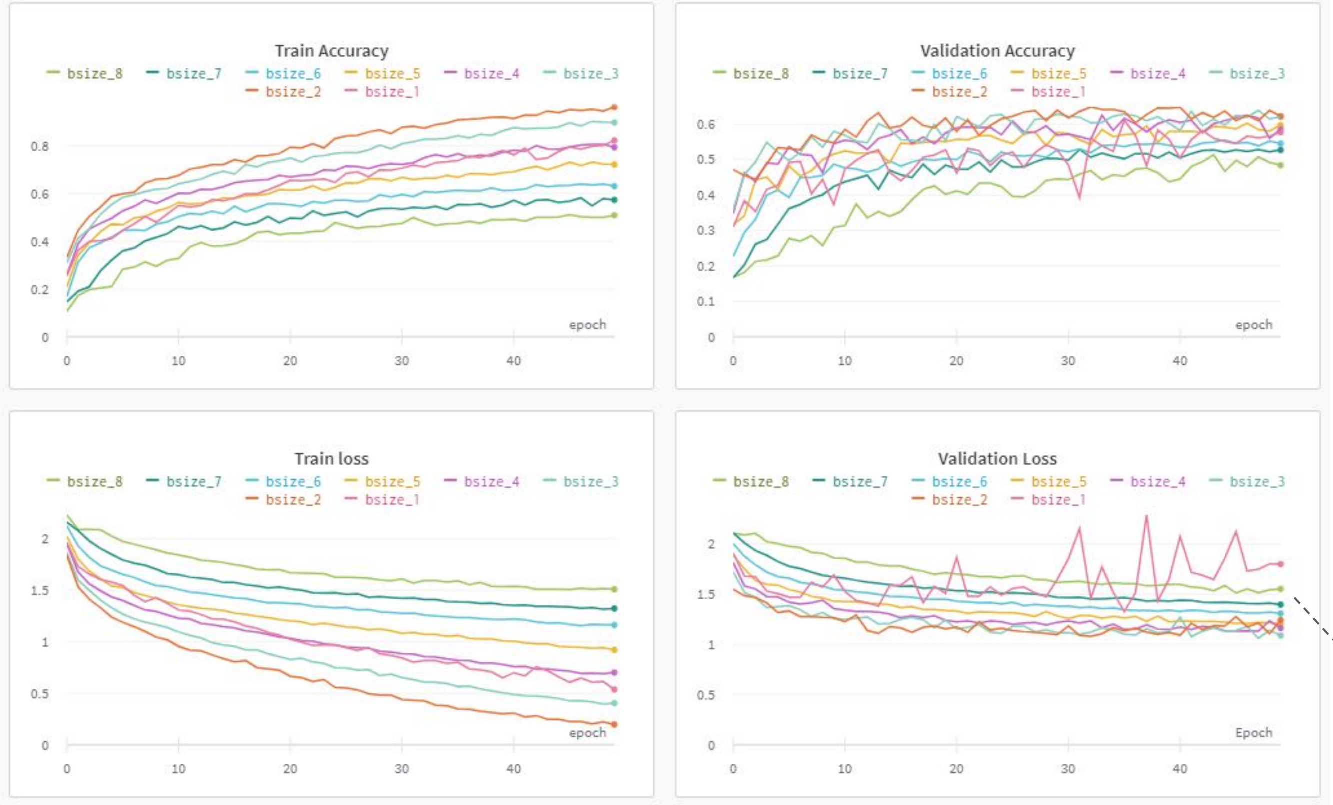


Image size

Isize_1 = 8x8
Isize_2 = 16x16
Isize_3 = 32x32
Isize_4 = 64x64
Isize_5 = 128x128

We don't gain validation accuracy augmenting the default Image size (32x32), and in fact, it is the one with the lower validation loss

Task 1: Changing the batch size



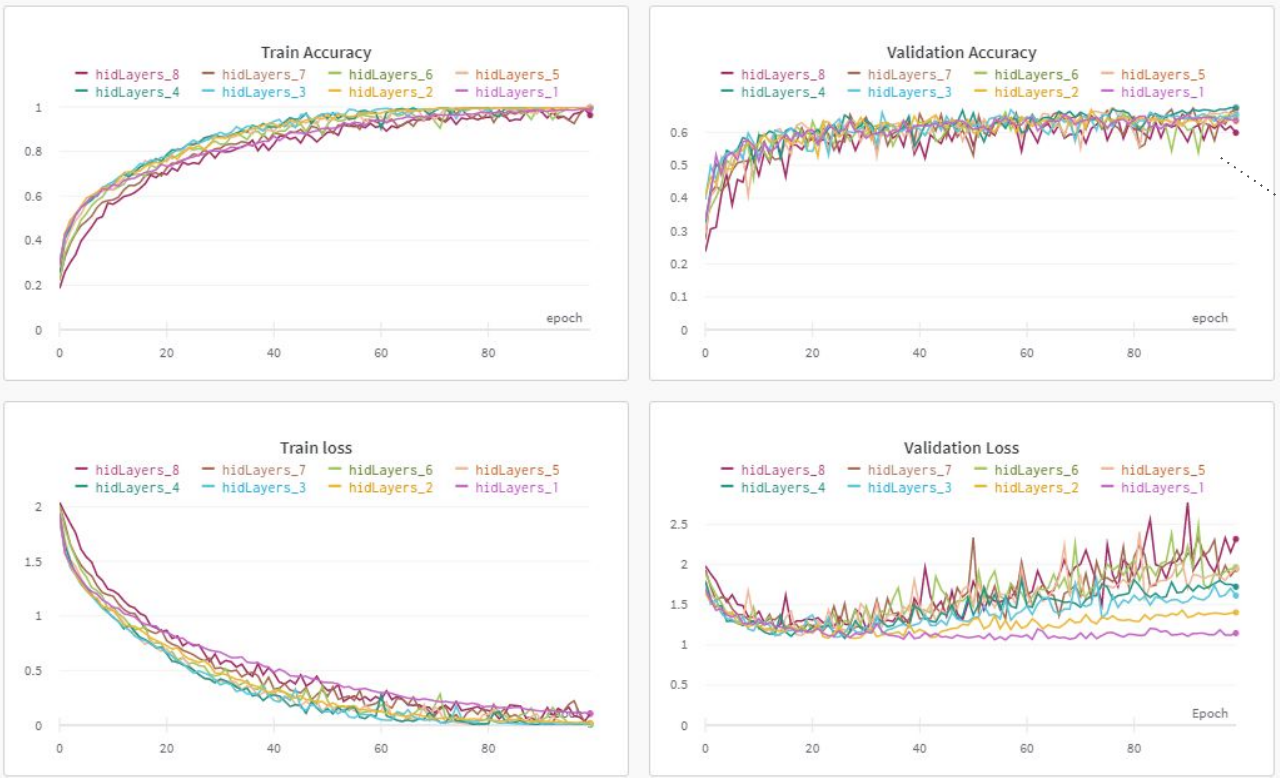
Batch size

bsize_1 = 1
bsize_2 = 8
bsize_3 = 16
bsize_4 = 32
bsize_5 = 64
bsize_6 = 128
bsize_7 = 256
bsize_8 = 512

The default batch size = 16 is appropriate, we have one of the highest accuracies and the validation loss is one of the lowest ones.

Note that smaller batches produce more unstable results, while larger batches improve the stability.

Task 1: Changing number of hidden layers



We start with the initial configuration of 1 Dense hidden layer with output 2048 (Model hidLayers_1) and sequentially add more dense hidden layers of half the size of the previous layer.

0.6762 hidLayers_4
0.6712 hidLayers_7
0.6563 hidLayers_5
0.655 hidLayers_3
0.6475 hidLayers_6
0.6363 hidLayers_1
0.6338 hidLayers_2
0.5987 hidLayers_8

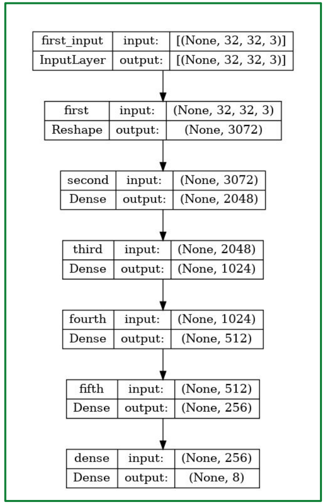
Adding some hidden layers improves the accuracy but also increases the overfitting (Validation losses start increasing).

With 4 hidden layers (see model in next slide) we obtain the best accuracy and the overfitting is not that bad as with more hidden layers. From now on, we will refer to this models as **model_1.4**

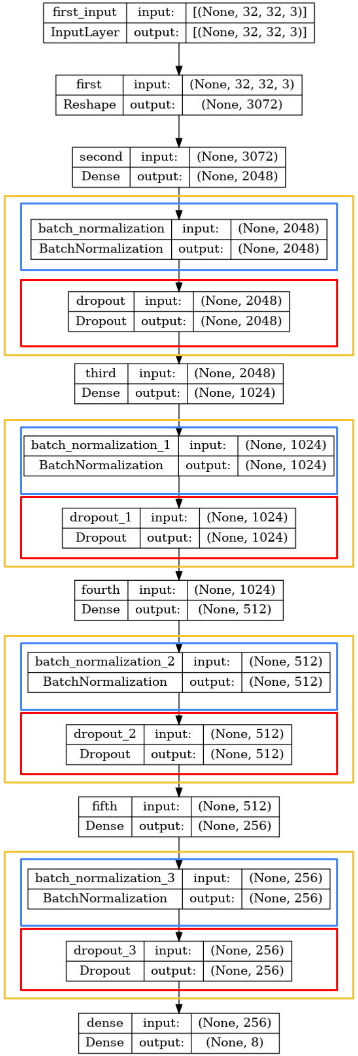
Task 1: Batch normalization layers and Drop out

Using the best model of the previous slide (model 1.4), we will add the following layers to try to reduce the overfitting:

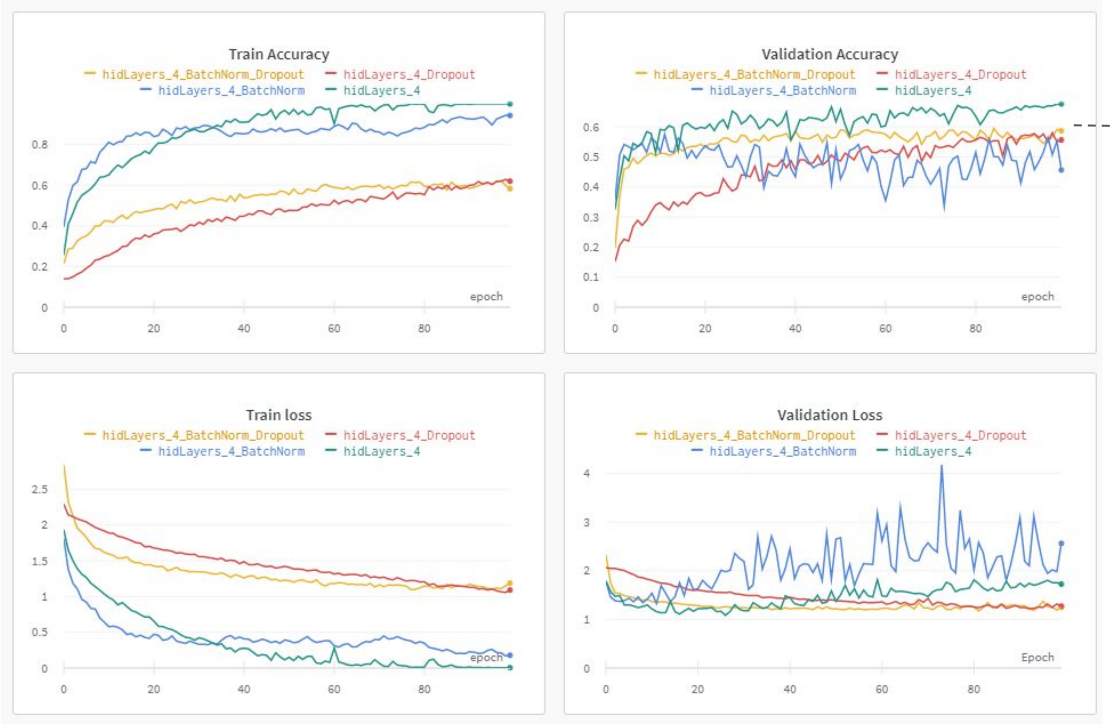
- Batch normalization layers after each dense hidden layer → Model 1.4.2
- Drop out layers after each dense hidden layer → Model 1.4.3
- Both, batch normalization and drop out layers after each dense layer → Model 1.4.4



Architecture of model 1.4

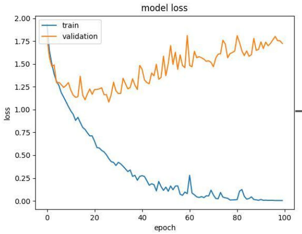


Task 1: Batch normalization layers and Drop out

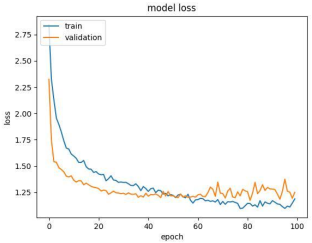


0.6762 hidLayers_4 → Model 1.4
0.5863 hidLayers_4_BatchNorm_Dropout → Model 1.4.4
0.5562 hidLayers_4_Dropout → Model 1.4.3
0.4563 hidLayers_4_BatchNorm → Model 1.4.2

Using the batch normalization and drop out layers, have reduced the overfitting and the gap between Train and Validation Losses.



Model 1.4



Model 1.4.4

However now the accuracy is 0.586 instead of 0.676

Task 1: Bigger layers

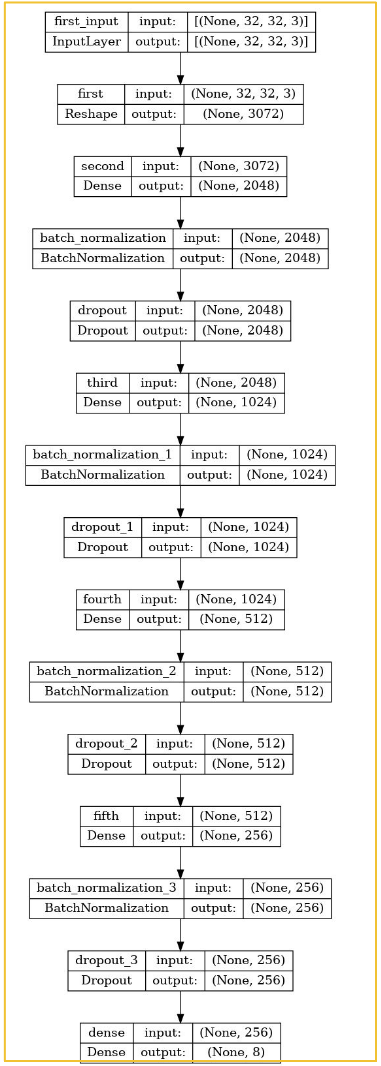
We will continue working with 4 hidden layers + Batch norm and drop out. However, now the first hidden layer output will be **4096** instead of 2048. We will also test having hidden layers of the same size.

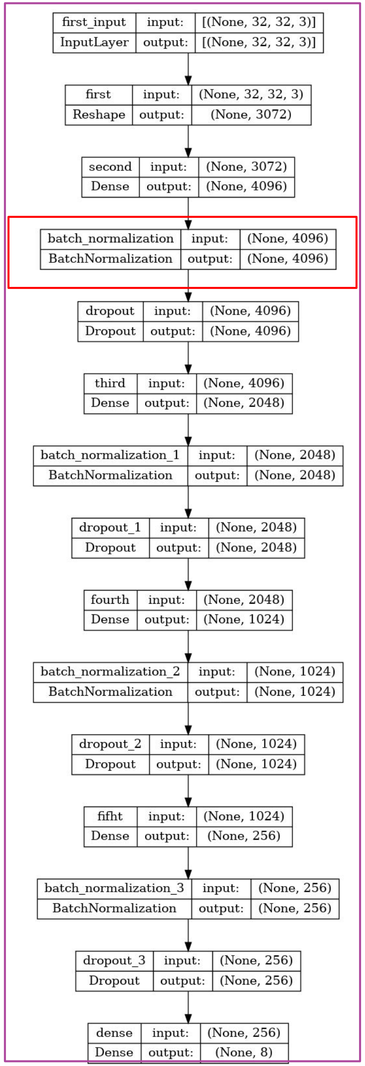
Model_1.4.4 → **9M** parameters

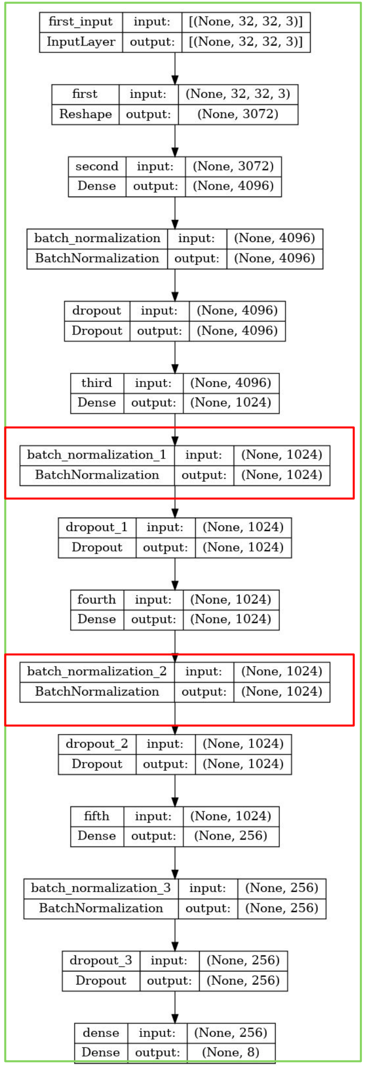
Model2 → **23M** parameters

Model3 → **18M** parameters

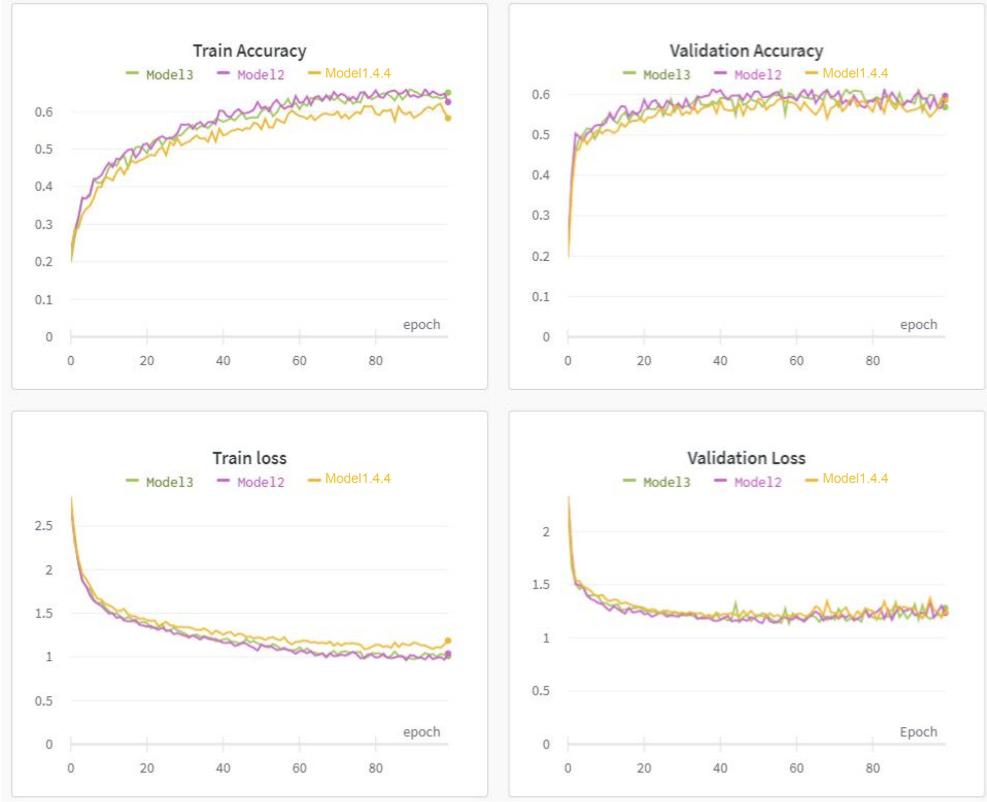
*We also tried to put only the last dropout layer instead of one for each hidden layer, but worst results were obtained







Task 1: Bigger layers



0.596 Model2
0. 5863 Model1.4.4
0.596 Model3

We see that starting with a bigger hidden layer slightly improves the validation accuracy and lowers a little bit the losses.

However, there are also more parameters to train (**9M** param model 1.4.4 → **23M** model 2).

Task 1: Changing the layer weight initializers

* By default the GlorotNormal class was being used, which draws samples from a truncated normal distribution .

```
0.5962 GarotNormal
0.58 TruncatedNormal
0.5638 RandomNormal
0.5375 RandomUniform
```

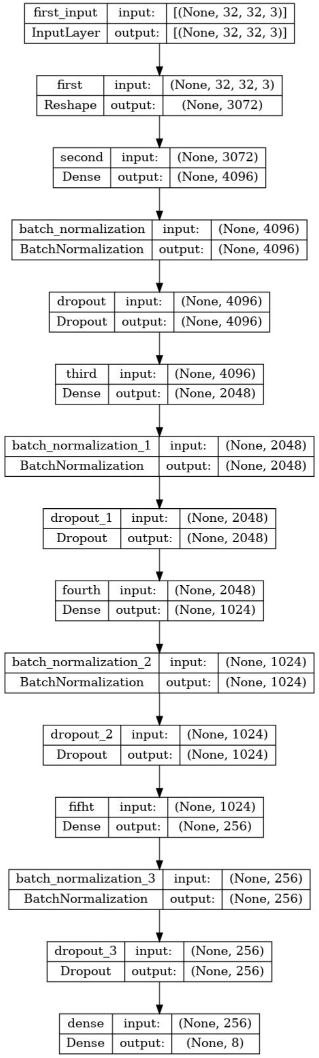
We see that the default option is the more appropriate, so we will not change it.

Task 1: Summary

We first tried to optimize the hyperparameters like the learning rate, batch size, image size… Then we tested different architectures using different number of hidden layers, adding batch normalization and dropout. The following table shows a summary of the architectures tested.

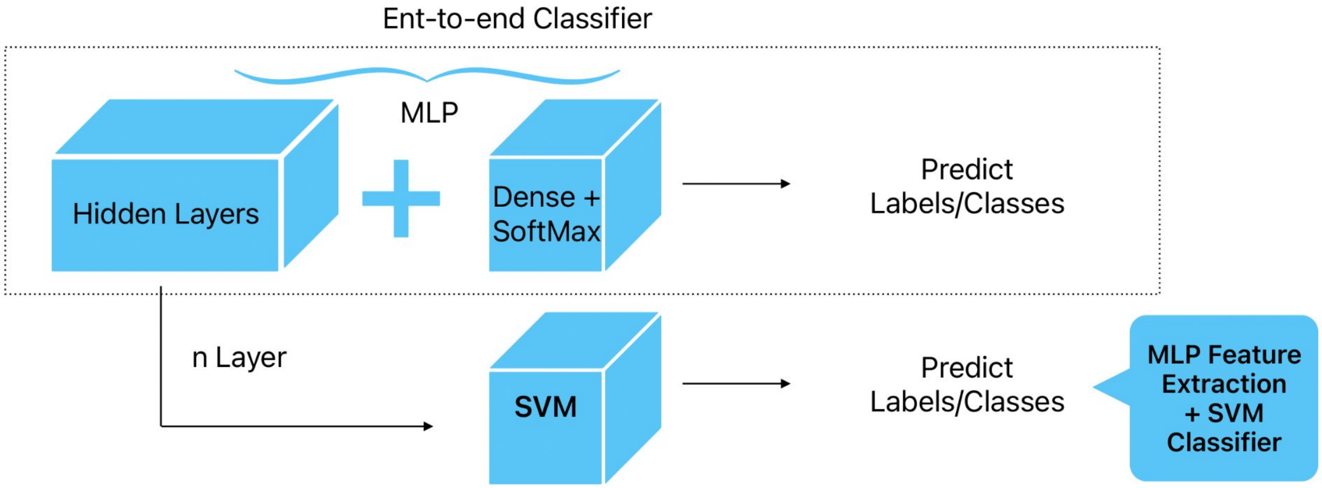
Name	Comments	Num of Parameters	train_loss	val_loss	train_accuracy	val_accuracy
Initial model	1 hidden layer with output 2048	6M	0.3875	1.123	0.9008	0.61
model_1.1	Initial model (with 100 epochs)	6M	0.1049	1.147	0.9903	0.6363
model_1.2	Initial model + 2 hidden layers	8M	0.01995	1.404	0.9995	0.6338
model_1.3	Initial model + 3 hidden layer	9M	0.007708	1.615	0.9995	0.655
model_1.4	Initial model + 4 hidden layers	9M	0.003655	1.725	0.9995	0.6762
model_1.5	Initial model + 5 hidden layer	9M	0.007779	1.949	0.9989	0.6563
model_1.6	Initial model + 6 hidden layers	9M	0.005447	1.967	0.9989	0.6475
model_1.7	Initial model + 7 hidden layer	9M	0.01388	1.945	0.9973	0.6712
model_1.8	Initial model + 8 hidden layers	9M	0.1054	2.319	0.963	0.5987
model_1.4.2	model_1.4 + BarchNorm after each hidden layer	9M	0.1775	2.561	0.9426	0.4563
model_1.4.3	model_1.4 + Dropout after each hidden layer	9M	1.089	1.272	0.6193	0.5562
model_1.4.4	model_1.4 + Barchnorm and Dropout after each hidden layer	9M	1.187	1.25	0.5828	0.5863
model 2	**model_1.4 with frist hidden layer output to 4096**	**23M**	**1.039**	**1.234**	**0.626**	**0.596**
model 3	Change first hidden layer output to 4096 + two hidden layers of the same size	18M	1.009	1.281	0.651	0.569

So, all in all, we conclude that the best model is model 2, since we reduced the gap between training and validation and reduced the overfitting. So, for next task, we will use model 2.



Task 2: Deep features + SVM

The second part is use MLP for extracting features and then train a SVM classifier. We use the kernel that optimizes the results of Week2 : “RBF”.



Task 2: Deep features + SVM

We try to find the different behaviors of extracting features from the best architecture MLP from Experiments 1 in different layers: second, third, fourth, fifth.

Architecture	Accuracy
Second Layer	**0.410**
Third Layer	0.408
Fourth Layer	0.400
Fifth Layer	0.406

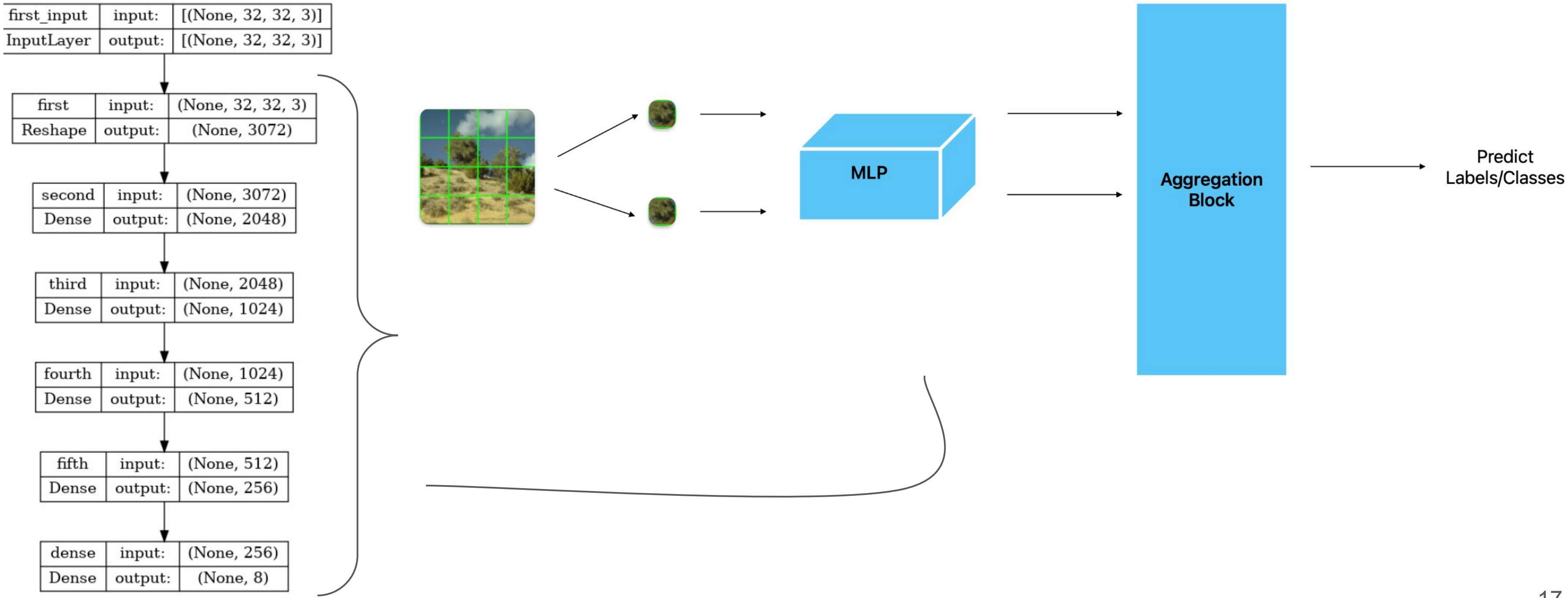
Task 1 accuracy: 0.596

Results are worse than the previous task, as expected.

The results barely change when taking different layers as outputs

Task 3: Patch based MLP

The third task consists on dividing the image in patches and for each one use MLP to classify it. Finally an aggregation block is used to classify the image. We will use the network of **model_1.4** based on 4 hidden layers.



Task 3: Patch based MLP

Here we perform two sets of experiments. The first one corresponds on finding the best patch size and the second one corresponds on changing the aggregation block using different functions (mean, max & min).

Patch Size (using mean)	Accuracy
8x8	0.623
16x16	0.705
32x32	**0.770**
64x64	0.757

Optimal patch size is 32x32

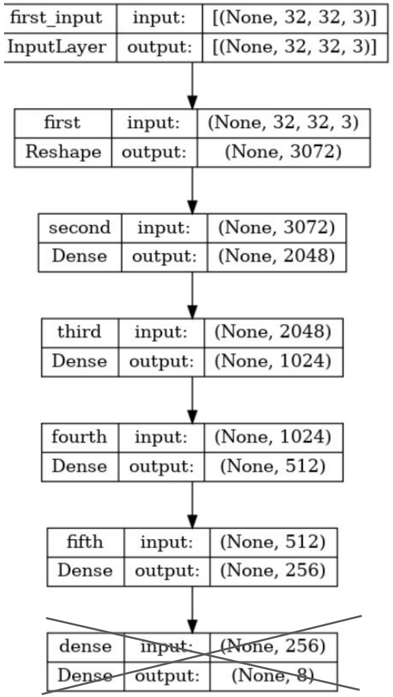
Decreasing the patch size below 32x32 leads to a worse classification (descriptors are too specific)

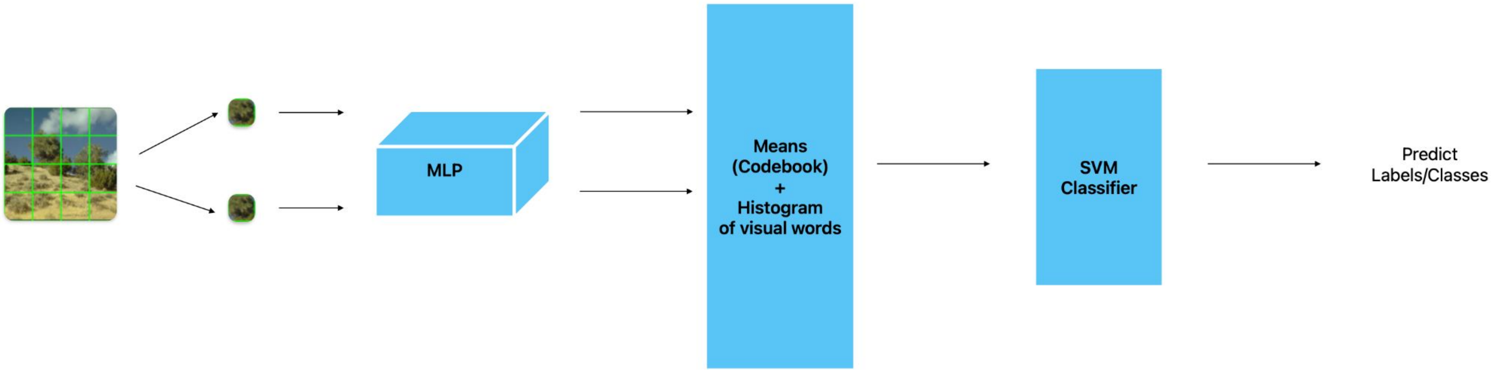
Aggregation statistics (using 32x32 patch size)	Accuracy
mean	**0.770**
max	0.564
min	0.524

Using **max** and **min** functions to aggregate channels decrease the accuracy with respect to the **mean** function

Task 4: Patch based deep features + BoVW

The fourth task consists on extracting the deep features from the last hidden layer (previous to softmax classification) for each image patch, and concatenate these features to create a feature vector for each image. Then, we use KMeans to create a codebook and we train an SVM classifier with the histograms of visual words:





Task 4: Patch based deep features + BoVW

In this last task we perform one set of experiments. It corresponds on finding the optimal number of clusters (the codebook size).

Number of clusters	Accuracy
64	0.691
128	0.701
256	0.726
512	0.678
1028	0.435

The best results are obtained with a codebook size of 256.

In this case, the end-to-end approach performs better than BoVW.

If the number of clusters is too low, there is too much generalization of the features.

If the number of clusters is too big, there is too much specificity of the features.

Conclusions

- Adding more layers improves the neural network.
- It seems like the depth of the network is more important to achieve better results than the breadth of the model (number of neurons).
- Regularization techniques such as batch normalization and dropouts help to make the network more robust and avoid overfitting.
- Excessive dropout causes the network not to learn.
- SVM strategy does not improve our results
- Divide each image in patches and extract deep features from each of them provides better results.
- Using these dense descriptors to create a Bag of Visual Words (BoVW) provides similar but slightly worse results.