MCV-M3: Image Classification (week 5)

Jonatan Poveda, Martí Cobos

Image Classification Pipeline

Non-Deep learning approach

Input Images and Labels

Feature Extraction

SIFT descriptor

Global image representation

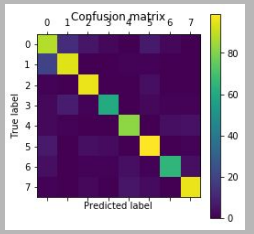
BoVW framework

Classify Image

SVM classifier

Evaluate Code Performance

Compute metrics using ground truth information



Deep learning approach

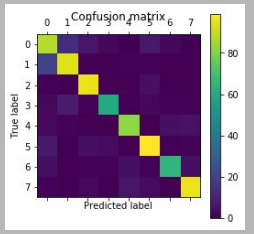
Input Images and Labels

Neural Network

Neural networks (VGG-16, CNN)

Evaluate Code Performance

Compute metrics using ground truth information



Week 1: Image Classification Pipeline

Train Database

Input Images and Labels

Feature Extraction

SIFT descriptor

Save Features

Save Features to test several Classifiers

Classifier Training

K-NN

Classification model

Test Database

Input Images and Labels

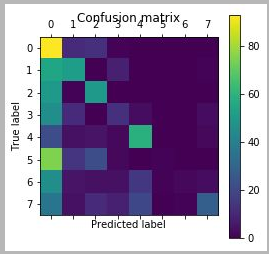
Feature Extraction

Classify Image

Feature extraction (SIFT) and classification

Evaluate Code Performance

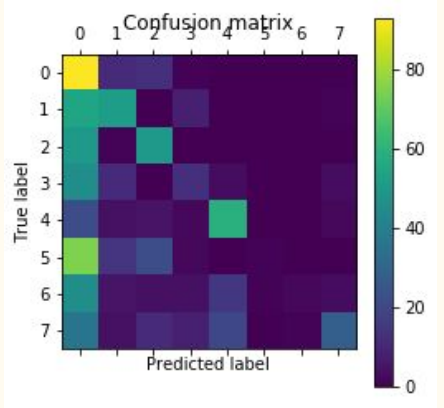
Compute metrics using ground truth



Week 1: Implement performance evaluation measures and experimental protocols

Performance evaluation metrics:

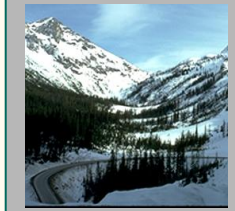
- Evaluator class has been implemented to assess code performance after assessing test images. Implementation based on sklearn.metrics class.
- Code performance metrics implemented:
  - Confusion matrix: Great for visualize similar classes on the feature space in use.
  - Accuracy; Metric used to assess correct classifications vs entire dataset
  - Precision & Recall
  - F-score: Metric used to assess overall classifier performance
- Image Classification algorithm evaluated on every code iteration



Week 1: Global description of the image

Color Histogram

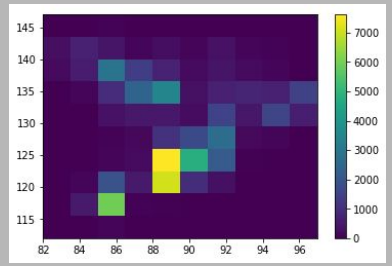
- A global feature descriptor based on the image color histogram has been implemented. Luv color space used for color histogram.
- Number of descriptors per image = (number of bins)^2
- The following flowchart summarizes the process to obtain the color histogram descriptor:



Input Images

Convert image to Luv

Compute 2D histogram of u, v



Convert matrix to 1D array

Week 1: Classifiers testing

- A Classifier builder has been implemented to select between the following classifiers:
  - K-NN
  - Random Forest
  - Gaussian & Bernoulli Bayesian Classifier
  - SVM
- A custom implementation of the logistic regression classifier has been implemented with the following features:
  - Convergence detection
  - Selectable maximum number of iterations
  - Lasso regularization with selectable lambda parameter
  - One vs All multiclass classification
- The following equation defines the cost function of the gradient descent algorithm implemented:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

Week 2: Image Classification Pipeline

Train Database

Input Images and Labels

Feature Extraction

SIFT descriptor

Vocabulary Learning

k-means clustering

Global image representation

BoVW framework

Classifier Training

SVM classifier

Classification model

Test Database

Input Images and Labels

Feature Extraction

SIFT descriptor

Global image representation

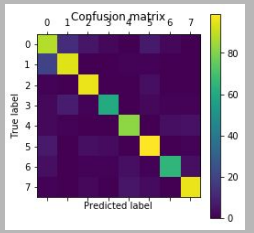
BoVW framework

Classify Image

SVM classifier

Evaluate Code Performance

Compute metrics using ground truth information

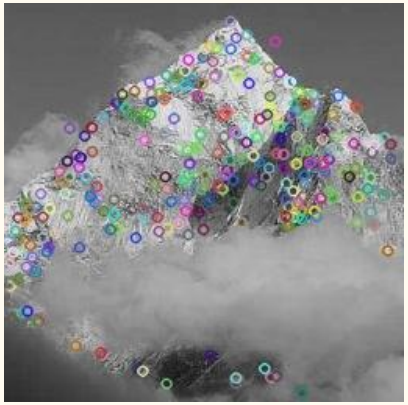


Week 2: Implement dense SIFT

Dense SIFT:

- Normal SIFT implementation computes features on corner detections. (See right image).
- Dense SIFT implemented by selecting key points uniformly spaced on the image. Python implementation using cv2.FeatureDetector_create("Dense") function
- Use standard SIFT descriptor to obtain the features on the selected key points.
- Dense SIFT improves accuracy from .66 to .79 (cross-validation average).

Standard SIFT:



Input Images

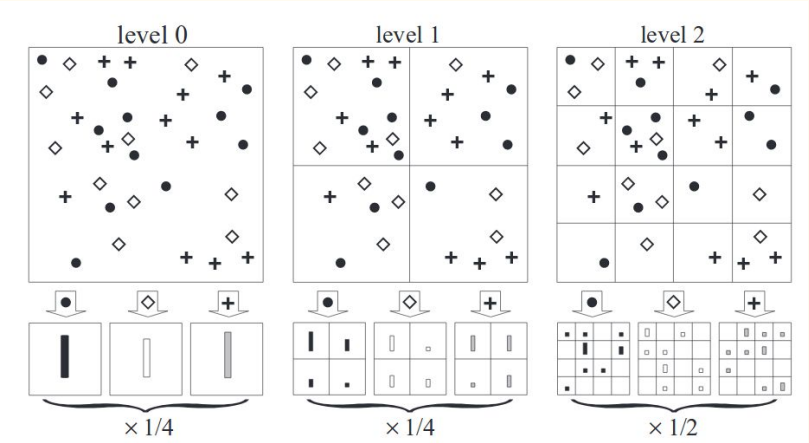
Keypoints uniform definition

SIFT descriptor computation

Image classification

Week 2: Spatial Pyramid and Histogram Intersection kernel

Spatial pyramids



Histogram Intersection

$$K_{int}(A, B) = \sum_{i=1}^{m} \min\{a_i, b_i\}.$$

Results

Pixel based	Accuracy	Precision	Recall	F1-score	time[s]
Method 1	.793	.799	.793	.796	579.2
Method 2	.844	.853	.844	.849	1104.3

- Method 1: Dense SIFT + SVM (RBF kernel)
- Method 2: Dense SIFT + Spatial Pyramids + SVM (histogram intersection kernel)

Weeks 3-5: Image Classification Pipeline

Implement a Multi Layer perceptron

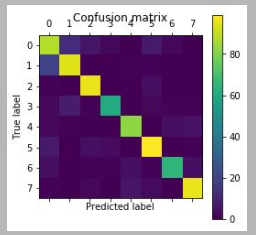
Input Images and Labels

Neural Network

Neural Network

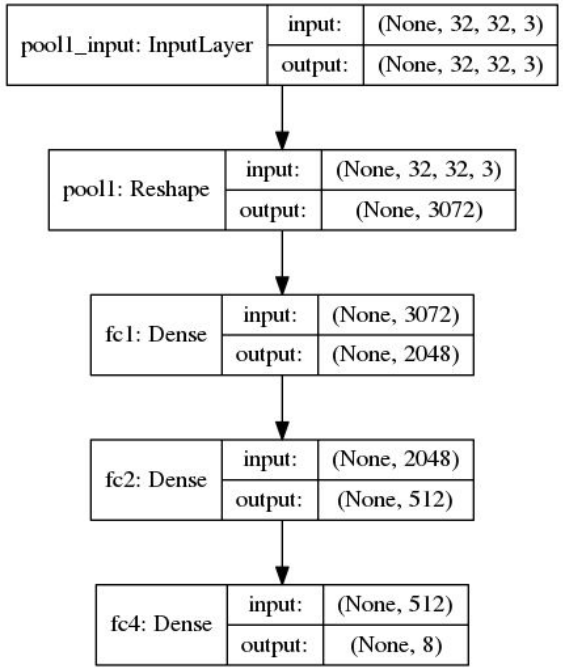
Evaluate Code Performance

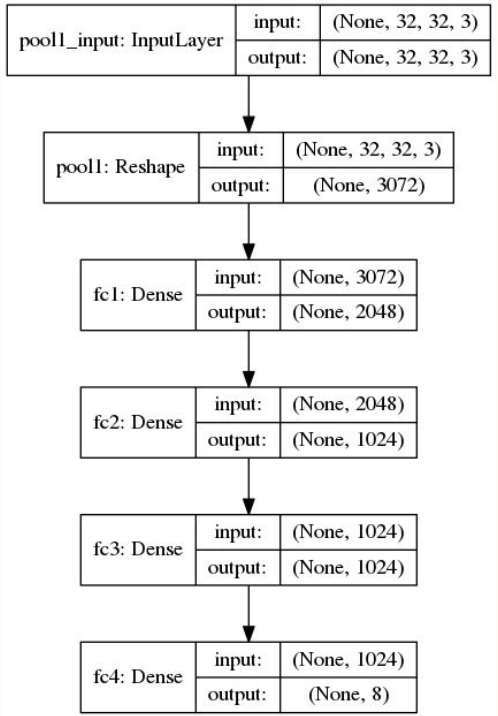
Compute metrics using ground truth information

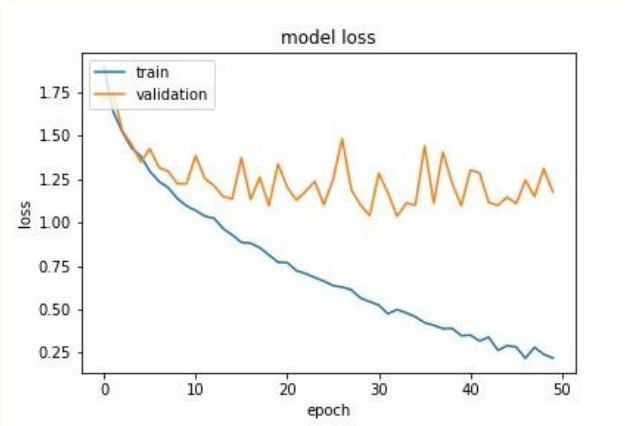


Week 3: Add/change layers in the network topology

- Several topologies have been tested by adding and changing layers on the base network architecture. The best topologies are shown below. Full evaluation results shown on slide 8







Model loss: there is no loss train convergence with 50 epochs. The architecture of the model should be modified

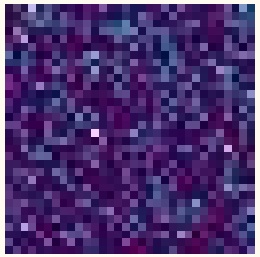
Week 3: Given an image, get the output of a given layer

Input image

MLP Topology

pool1_input: InputLayer	input:	(None, 32, 32, 3)
	output:	(None, 32, 32, 3)
pool1: Reshape	input:	(None, 32, 32, 3)
	output:	(None, 3072)
fc1: Dense	input:	(None, 3072)
	output:	(None, 2048)
fc2: Dense	input:	(None, 2048)
	output:	(None, 1024)
fc3: Dense	input:	(None, 1024)
	output:	(None, 1024)
fc4: Dense	input:	(None, 1024)
	output:	(None, 8)

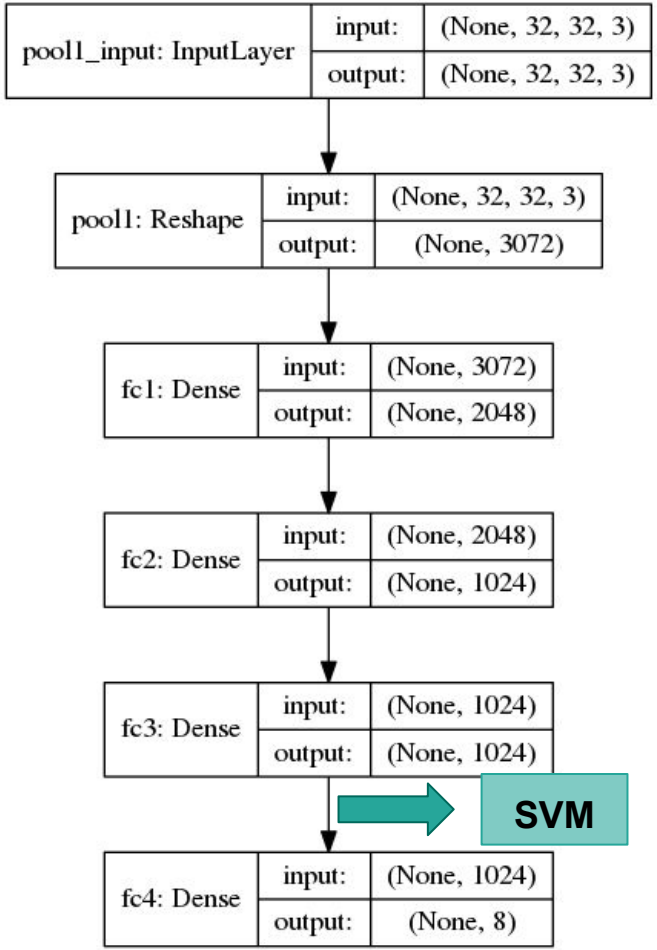
MLP output is difficult to interpret using only fully connected layers.



Output image

Reshape array to 32x32 image

Week 3: Extract a single feature from an input and apply to SVM



Procedure to apply SVM to the MLP output

1. Train multi layer perceptron with train images
2. Get the image output from the last hidden layer
3. Cross-validate SVM with output values to find best SVM kernel and parameters (best kernel: linear)
4. Train SVM with train images
5. Predict test images and evaluate classifiers

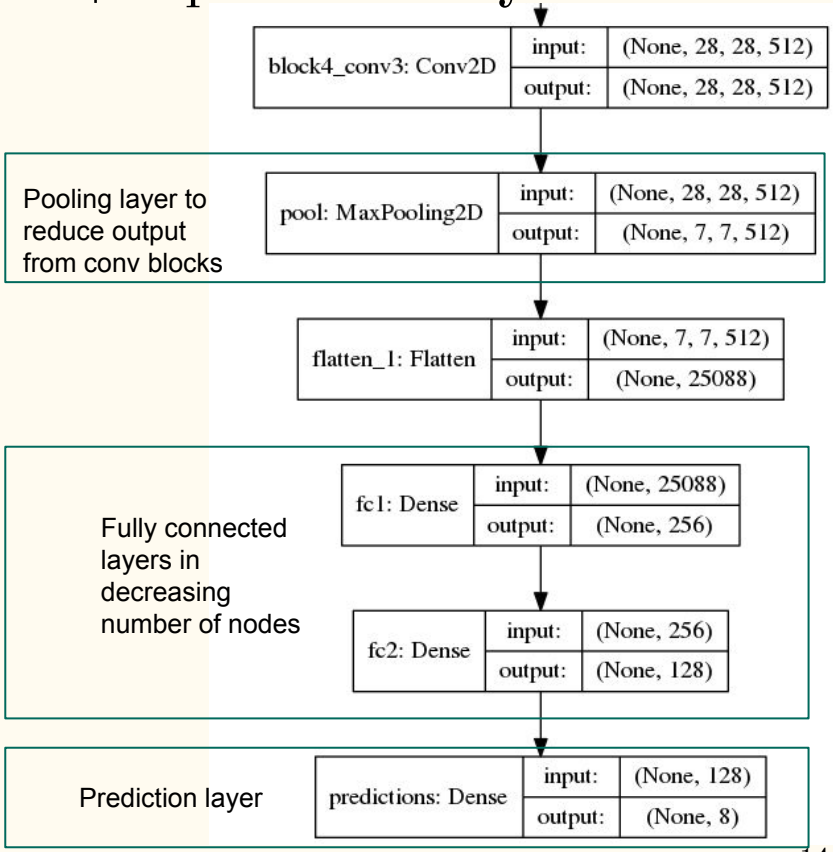
Evaluation results on test images:

Image based	Accuracy	Precision	Recall	F1-score	time[s]
MLP	.640	.640	.657	.646	1122.0
SVM	.660	.674	.667	.670	1141.8

SVM results are slightly better than end to end MLP

Week 4: Set a new model from a layer below VGG16-block4 including at least a fully connected layer + a prediction layer

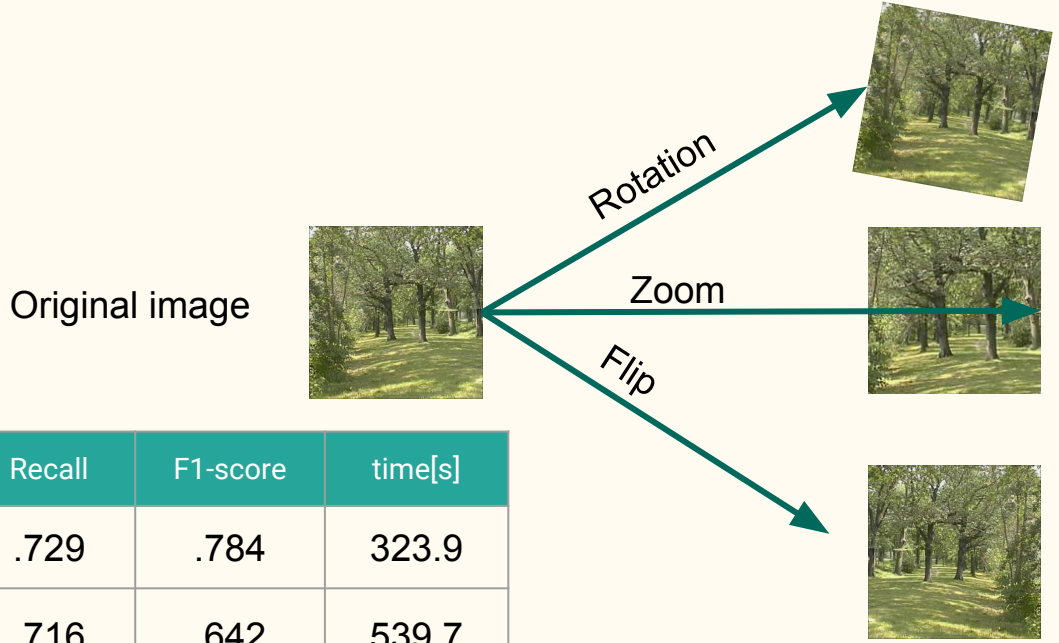
- The last convolutional blocks of VGG 16 is eliminated. A pooling layer to reduce the number of outputs from the convolutional blocks, two fully connected layers and a softmax layer are added at the end of the third block as shown on right image.
- The neural network is trained in 2 stages:
  - 1st stage: Freeze layers until the end of block 3, and train the new fully connected layers.
  - 2nd stage: Unfreeze all layers to train all the CNN.
- The learning rate on the 2nd training stage is reduced in order to avoid accuracy drop when changing the pre-trained layers.



Week 4: Introduce and evaluate the usage of data augmentation

In order to increase the number of train images, using a base of only 400 real images, data augmentation has been implemented. The following techniques have been implemented:

- Random rotation ($\pm 15^{o}$)
- Random width shift (90%)
- Random zoom (30%)
- Random horizontal flip
- Filling with reflection

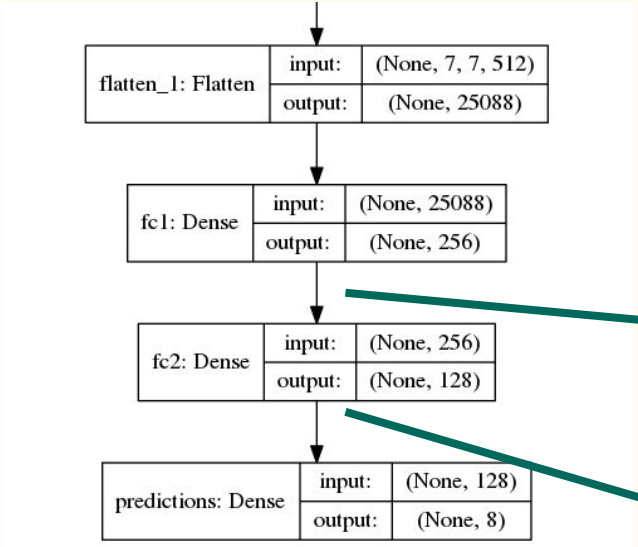


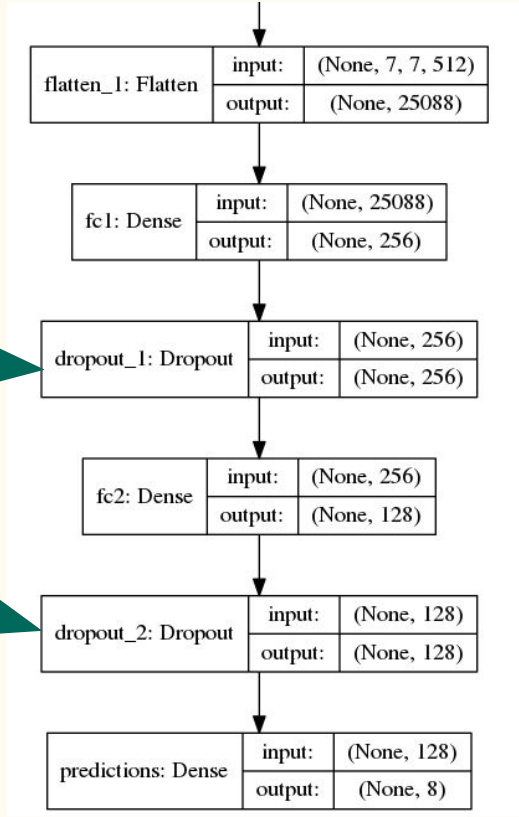
Results on test image set:

Image based	Accuracy	Precision	Recall	F1-score	time[s]
Normal	.747	.780	.729	.784	323.9
With data augmentation	.674	.639	.716	.642	539.7

Week 4: Introduce and evaluate the usage of a dropout layer

- A dropout layer is added after the last 2 fully connected layers:





Results on test image set:

Image based	Accuracy	Precision	Recall	F1-score	time[s]
With data augmentation	.674	.639	.716	.642	539.7
Dropout Added	.489	.465	.403	.461	521.6

Week 4: Apply random search on per model hyperparameters

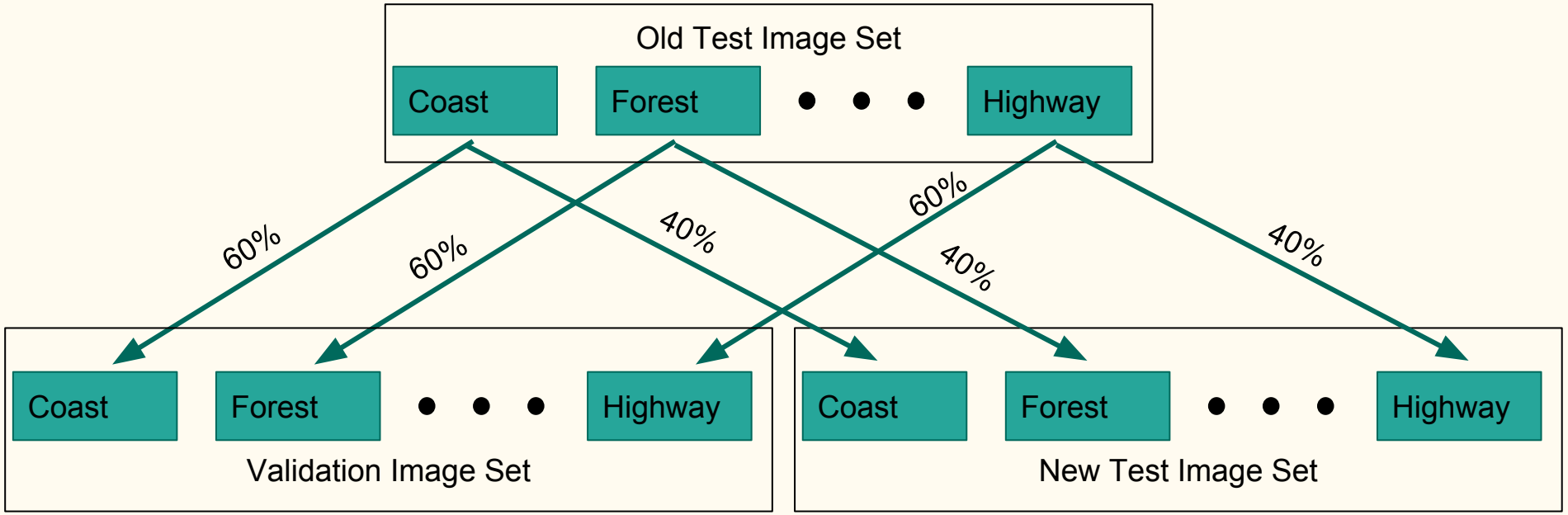
- We had troubles in order to adapt the initial CNN. By the time those issues were fixed, we could not perform a large random search of the hyperparameters due to a lack of time.
- The following parameters were added to the random search:
  - Batch size (possible values: 16, 32, 64)
  - Fully connected layer 1 size (possible values: 512, 1024, 2048, 4096)
  - Fully connected layer 2 size (possible values: 128, 256, 512, 1024)
  - (We didn't try to search over the Adadelta optimize parameters because it was discouraged by Keras library)
- Best results were achieved with a batch size of 32 and the size of the fully connected layers set to 1024. Evaluation results over the training set are shown on the following table:

Image based	Accuracy	Precision	Recall	F1-score	time[s]
Best random search results	.557	.467	.580	.466	292

Note: Random search did not improve previous due to lack of time to implement a more exhaustive search

Week 5: Test/Validation split

- Test image set provided has been splitted in a new test and a validation dataset with a 60% of images moved to validation set:



- During CNN training, the validation is used to tune the hyper-parameters. Afterwards, the image classification algorithm performance is assessed with test image set.

Week 5: Custom model

- We build some models from scratch, smaller than VGG16 due to we need only to classify over 8-classes instead of 100-classes.
- A Bayesian Optimization was implemented to find the best hyper-parameters:
  - 3 different architectures were evaluated
  - 3 image sizes (64x64, 128x128 and 256x256 [original]) as a input
  - a batch size of 128 images, ~300 iterations per epoch
  - Adadelta as the optimizer with recomm. parameters (lr=1, rho=.95)
  - a categorical cross-entropy as a loss function
  - accuracy as the metric to maximize
  - only 20 epochs of training

Week 5: Custom model

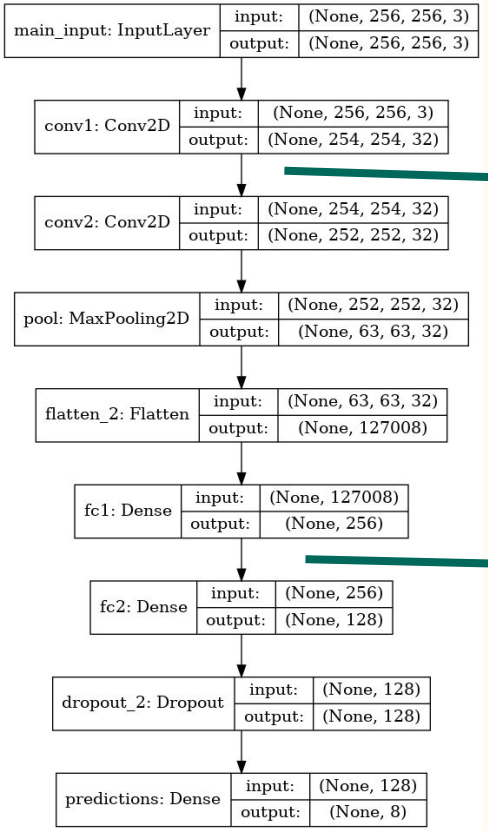
- Arch-1 is based on two conv. layers and a pooling with three fc layers with one dropout before the layer of predictions.

- Arch-2 is a modification of Arch-1 that reduces and number of parameters by adding a pooling layer after each conv., in that way we expect to keep the most representative filters and adds a bit of location invariance. It also has a dropout after each fc layer to enforce filters definition.

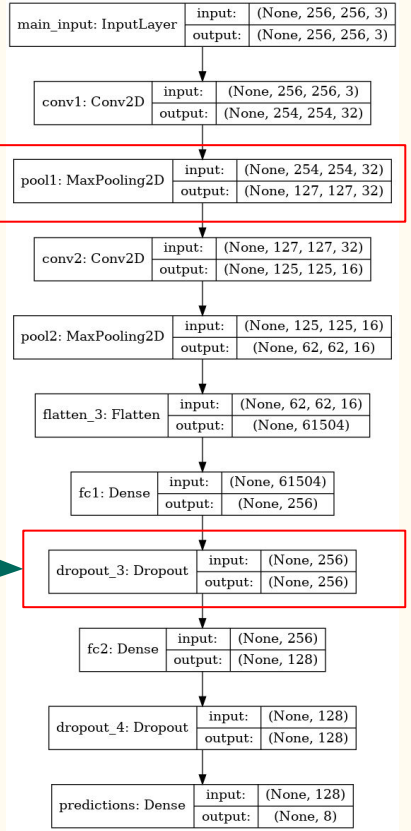
- Arch-3 is a modification of Arch-2 with twice filters in the conv. layers, expecting to retrieve more features both low-level and high-level ones.

Week 5: Evaluated architectures

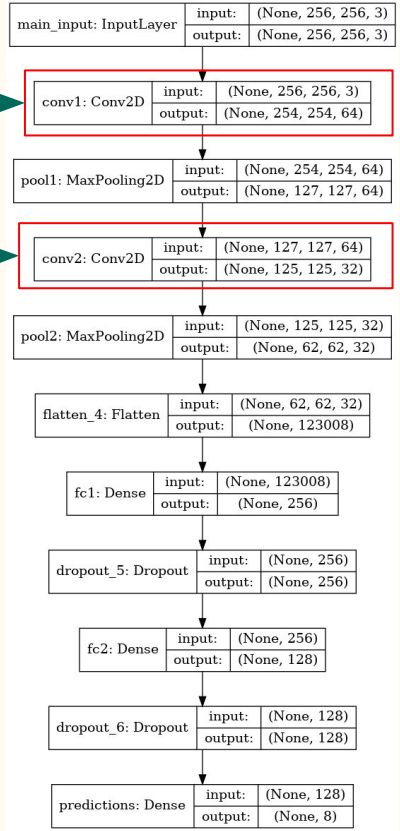
Arch-1



Arch-2



Arch-3



Week 5: Custom model

- Comparison

Image based	Accuracy	Precision	Recall	F1-score	# Param	time[s]
M3-I64-E50	**.853**	**.867**	**.846**	**.856**	**1.66 M**	**2917**
M3-I64-E20	.831	.844	.827	.836	1.66 M	1222
M3-I256-E50	.831	.837	.830	.834	31.54 M	2932
M3-I128-E50	.825	.834	.826	.830	7.43 M	4862
M2-I64-E50	.818	.831	.816	.823	0.84 M	5023
M2-I128-E20	.821	.829	.818	.823	3.72 M	2158
M2-I128-E50	.816	.831	.810	.820	3.73 M	5367
M1-128-E20	.806	.813	.804	.809	7.91 M	2060
M1-I256-E20	.796	.809	.797	.802	32.59 M	5257

Note: M{x}-I{y}-E{z} is the code for model {x} with an input image of {y}x{y} with {z} epochs

Classification performance evaluation (on test images)

Image based	Accuracy	Precision	Recall	F1-score	time[s]
Week 1	.265	.274	.267	.270	1.17
Week 2	.844	.853	.844	.849	1104
Week 3	.660	.674	.667	.670	2142
Week 4	.747	.780	.729	.784	324
Week 5	**.853**	**.867**	**.846**	**.856**	**2917**

Algorithm performance evaluated on test images using the following architectures (best results from every week):

- Week 1: ColorHist (32 bins) + SVM (C=1)
- Week 2: dense SIFT + k-means + spatial pyramids + SVM (histogram intersection kernel)
- Week 3: 3 hidden layer multi layer perceptron + SVM (task 4)
- Week 4: VGG 16 block4+4-pool+4096fc+1024fc
- Week 5: CNN M3-I64-E50

Conclusions

- BoVW is a good starting point for solving the classification problem due the low number of images. However, deep learning has a lot of potential and can achieve human accuracy.
- Applying deep learning can boost the performance of the classification. However more hyper-parameters (optimizer, initializer, architecture) should be taking into account to select the best model. In addition, the duration of the training phase is significantly higher, without GPU acceleration this process wouldn't have been possible in a reasonable time.
- Best results achieved using the CNN's architecture 3 using an image size of 64x64x3.
  - Counter-intuitively, a bigger image size seems to worse the results. Apart from increase a lot the number of parameters to train.
  - Adding an extra pooling and an extra dropout improve the results

Pixel based	Accuracy	Precision	Recall	F1-score	time[s]
Arch-{N}	**.853**	**.867**	**.846**	**.856**	**2917**

Future work, (using DL)

- Collect more training data
- Increase the number of epochs to further analyse the models and get a better working point
- Increase the search in hyper-p. space to find better ones
- Implement visualization to extract some info and fine-tune the NN
- Base our model in other well-known NN like Inception
- Give a try to the Capsule Networks which are based on CNNs
- Maybe other types of NNs...

Best method Confusion Matrix plot