[M2, MVA]

Object recognition and computer vision

Maha ELBAYAD
maha.elbayad@student.ecp.fr

Assignement 3

November 17, 2015

Part 1: Training a neural network by back propagation

Stage A: Computing gradients of the loss with respect to network parameters

QA- The model parameters are $W_o \in \mathbb{R}^{1\times h}$, $B_o \in \mathbb{R}$, $W_i \in \mathbb{R}^{h\times 2}$ and $B_i \in \mathbb{R}^h$, with h the number of hidden units.
Our input is $X \in \mathbb{R}^2$ and the output is $\bar{Y} = W_o H + B_o \in \mathbb{R}$, with $H = ReLU(W_i X + B_i)$
We are using the logistic loss:

$$s(Y, \bar{Y}(X)) = \log(1 + \exp(-Y.\bar{Y}(X)))$$

To compute the gradient we start with:

$$\frac{\partial s(Y, \bar{Y}(X))}{\partial \bar{Y}} = \frac{-Y \exp(-Y.\bar{Y})}{1 + \exp(-Y.\bar{Y})} = -\frac{Y}{1 + \exp(Y.\bar{Y})}$$

And then:

$$\frac{\partial \bar{Y}(X)}{\partial W_o} = H^T,\ \frac{\partial \bar{Y}(X)}{\partial B_o} = 1$$

$$\frac{\partial \bar{Y}(X)}{\partial W_i} = (W_o^T \odot \mathbb{1}[W_i X + B_i \succ \mathbf{0}_h]).X^T$$

$$\frac{\partial \bar{Y}(X)}{\partial B_i} = W_o^T \odot \mathbb{1}[W_i X + B_i \succ \mathbf{0}_h]$$

Where for a given vector $v \in \mathbb{R}^p$, $\mathbb{1}[v \succ \mathbf{0}_p] = (\mathbb{1}[v_1 > 0], ..., \mathbb{1}[v_p > 0])^T$ and $\odot$ denotes the elementwise multiplication.
Hence:

$$\frac{\partial s(Y, \bar{Y}(X))}{\partial W_o} = \frac{\partial s(Y, \bar{Y}(X))}{\partial \bar{Y}} . \frac{\partial \bar{Y}}{\partial W_0} = -\frac{Y}{1 + \exp(Y.\bar{Y})} . H^T$$

$$\frac{\partial s(Y,\bar{Y}(X))}{\partial B_o} = -\frac{Y}{1+\exp(Y.\bar{Y})}$$

$$\frac{\partial s(Y,\bar{Y}(X))}{\partial W_i} = -\frac{Y}{1+\exp(Y.\bar{Y})}.(W_o^T \odot \mathbb{1}[W_i X + B_i \succ \mathbf{0}]).X^T$$

$$\frac{\partial s(Y,\bar{Y}(X))}{\partial B_i} = -\frac{Y}{1+\exp(Y.\bar{Y})}.W_o^T \odot \mathbb{1}[W_i X + B_i \succ \mathbf{0}]$$

Stage B: Numerically verify the gradients

QB1- To verify our analytically computed gradients, we will numerically compute the approximate derivative of the loss $s(\Theta)$ w.r to Θ_i using finite differencing:
Assuming s is a scalar-valued function of a vector variable $s : \mathbb{R}^p \to \mathbb{R}$

$$s(\Theta + \Delta\Theta) \approx s(\Theta) + \Delta\Theta^T.\nabla s(\Theta) = s(\Theta) + \sum_i \Delta\Theta_i \frac{\partial s(\Theta)}{\partial \Theta_i}$$

To retrive $\frac{\partial s(\Theta)}{\partial \Theta_i}$ we set $\Delta\Theta = \epsilon e_i$ where $(e_i)_{1\leq i\leq p}$ is the canonical basis of $\mathbb{R}^p$.
Hence:

$$\frac{\partial s(\Theta)}{\partial \Theta_i} = \frac{s(\Theta + \epsilon e_i) - s(\Theta)}{\epsilon}$$

For more accurate verification we'll use:

$$\frac{\partial s(\Theta)}{\partial \Theta_i} = \frac{s(\Theta + \epsilon e_i) - s(\Theta - \epsilon e_i)}{2\epsilon}$$

QB2- For a randomly selected training example $\{X, Y\}$ we verify that the analytically computed gradients match the numerical ones and we compute the norm of their differences.

```
grad_s_Wi_approx =

    2.2480   -0.3206
    0.3672   -0.0524
    2.2613   -0.3225


grad_s_Wo_approx =

    1.8696    2.4402    1.2089


grad_s_bi_approx =
```

```
    0.9527
    0.1556
    0.9584


grad_s_bo_approx =

    0.9874


grad_s_Wi =

    2.2480   -0.3206
    0.3672   -0.0524
    2.2613   -0.3225


grad_s_Wo =

    1.8696    2.4402    1.2089


grad_s_bi =

    0.9527
    0.1556
    0.9584


grad_s_bo =

    0.9874

For X=[2.3595,-0.3365] of label Y=-1
Gradient etimation error on bo= 2.248e-11
Gradient etimation error on Wo= 3.380e-10
Gradient etimation error on bi= 2.521e-11
Gradient etimation error on Wi= 3.395e-10
```

Stage C: Training the network using backpropagation and experimenting with different parameters

QC1- We train our neural network of 7 hidden units with a learning rate of .02. The resulting decision boundary is shown in figure (1) as well as the evolution of both training and validation errors in figures (2a,2b). The final training error being 0% reached starting from 72717 passes through the samples.

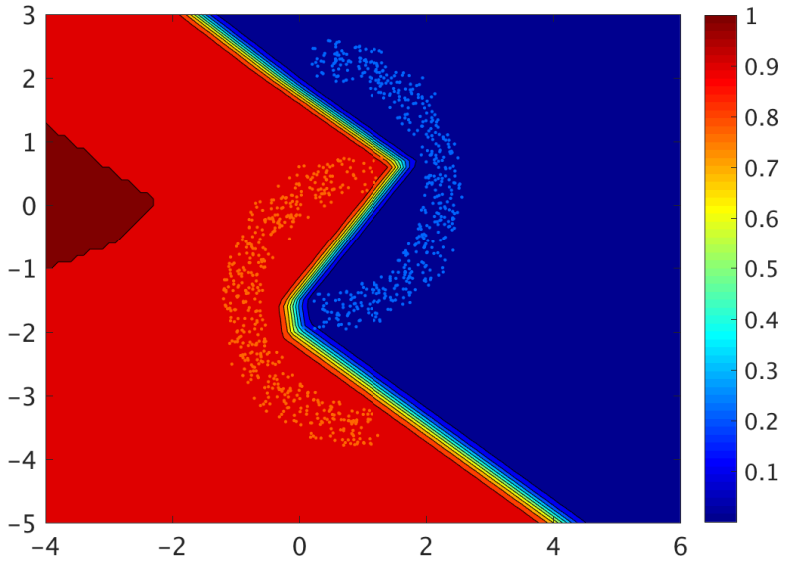
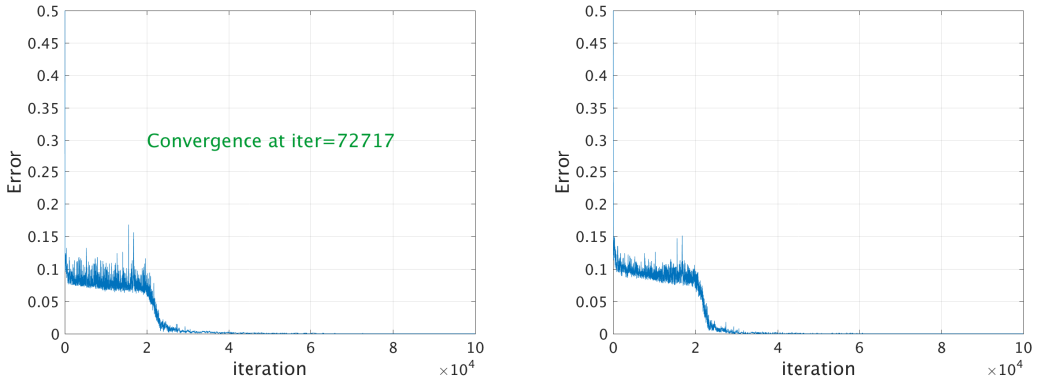


Figure 1: The neural net's decision boundary



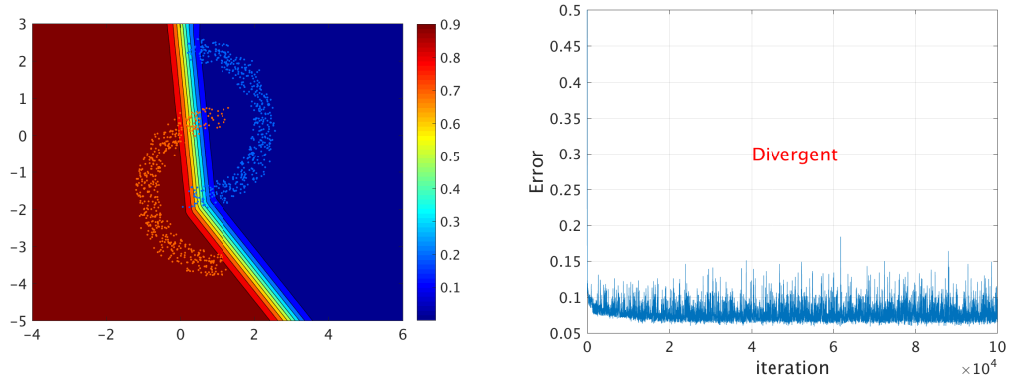
(a) Training error (b) Validation error

Figure 2: Convergent case

QC2- We run the training of our neural network 5 times starting from random initialization of the parameters $\{W_o, B_o, W_i, B_i\}$. The model did converge 4 out of 5 times at variant points (the limit being $100 \times n_{samples}$ passes). In the divergent case:

- Training error=7.30%
- Validation error=9.50%

The resulting decision boundary is shown in figure (3a)



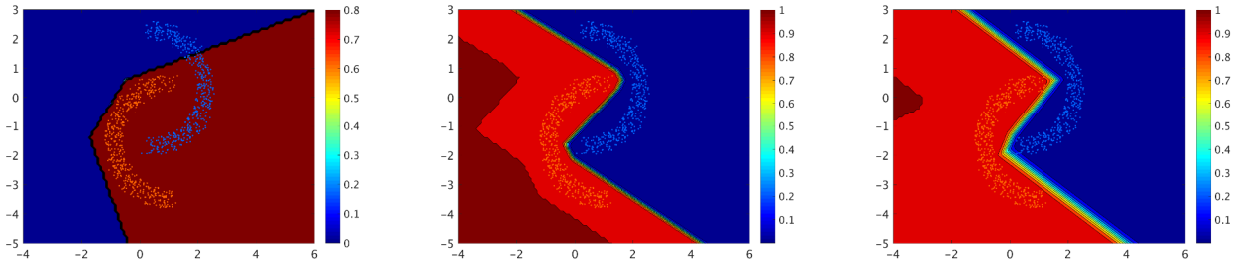
(a) The NN's decision boundary (b) Training error

Figure 3: Divergent case

QC3- We change the learning rate of the model taking consecutively the values $\{2, 0.2, 0.02, 0.002\}$ and run each case three times with random parameters initialization. The table (1) reports the statistics of each run[1].

Figure 4: Decision boundaries

(a) lrate=2 (b) lrate=.2 (c) lrate=.002



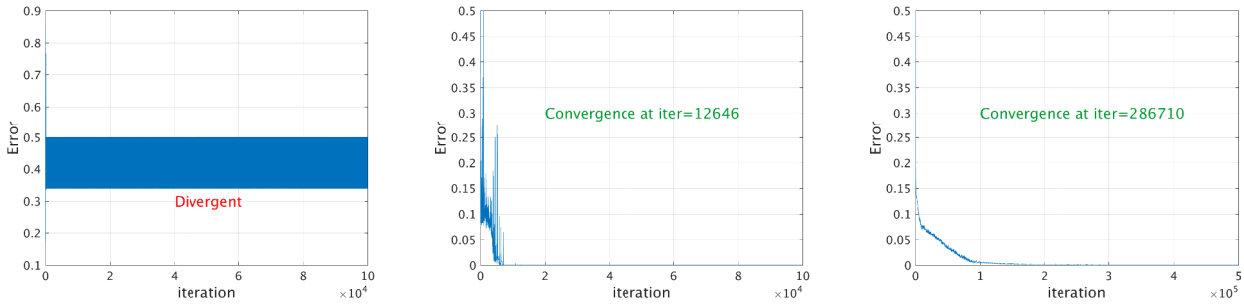
[1]Increased maxiumum passes to $500 \times n_{samples}$ for lrate=.002

lrate	run	Training error	Validation error	status
2	1*	33.90%	36.50%	Divergent
	2	36.00%	38.30%	Divergent
	3	26.10%	26.70%	Divergent
.2	1	0%	0%	CV iter=13804
	2	0%	0%	CV iter=12937
	3*	0%	0%	CV iter=23085
.02	1	7.50%	9.30%	Divergent
	2	0%	0%	CV iter=47271
	3	0%	0%	CV iter=51425
.002	1	8.20%	10.60%	Divergent
	2	0%	0%	CV iter=469957
	3*	0%	0%	CV iter=286710

Table 1: Different learning rates - statistics

Figure 5: Training errors

(a) lrate=2 (b) lrate=.2 (c) lrate=.002



For each lrate value (excpet the initial .02) we report an instance of the decision boundary and the training errors progress (the * run). In the case of $lrate = 2$ the learning rate is too large to permit convergence as it keeps bouncing away from the min value (figure 5a). When lrate is too small, the model (if convergent) takes too many iterations as the update towards is insignificant (figure 5c).

QC4- We set the learning rate to .02 and change the number of hidden units $h \in \{1, 2, 5, 7, 10, 100\}$. We run each setting three different times and report some of the results in table (2))

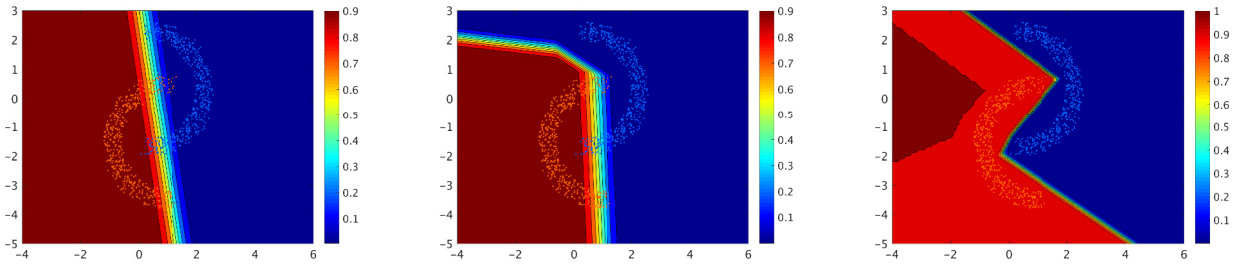
h	run	Training error	Validation error	status
1	1	8.90%	10.50%	Divergent
	2	8.30%	10.60%	Divergent
	3*	8.20%	10.7%	Divergent
2	1	7.80%	9.90%	Divergent
	2	9.80%	10.80%	Divergent
	3*	8.30%	9.40%	Divergent
5	1	0%	0%	CV iter=240895
	2	0%	0%	CV iter=57120
	3*	0%	0%	CV iter=45832
7	1	9%	7.70%	Divergent
	2	0%	0%	CV iter=63665
	3	0%	0%	CV iter=36509
10	1	0%	0%	CV iter=60273
	2	0%	0%	CV iter=51915
	3*	0%	0%	CV iter=53605
100	1	0%	0%	CV iter=33767
	2	0%	0%	CV iter=36513
	3*	0%	0%	CV iter=41602

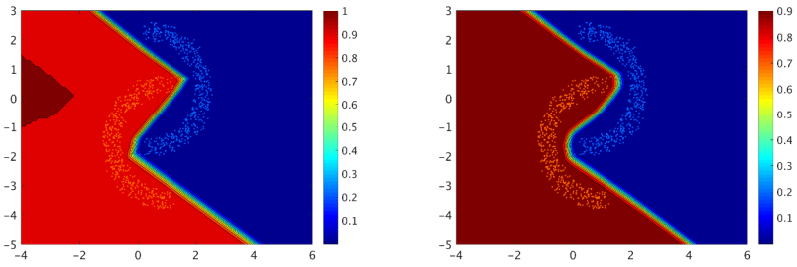
Table 2: Different learning rates - statistics

For each h value (excpet the initial 7) we report an instance of the decision boundary and the training errors progress (the * run)
When changing h we notice that small hidden units models don't converge as they have little freedom degrees to differntiate the two-moons: in fact the decision boundary with $h = 1$ is linear (figure 6a) and is quadratic with $h = 2$ (figure 6b) . Starting from h=5, the models converge and reach the optimal value faster (in number of iterations) with larger h. Moreover, a high value of h generates a smoother decision boundary.

Figure 6: Decision boundaries

(a) h=1 (b) h=2 (c) h=5

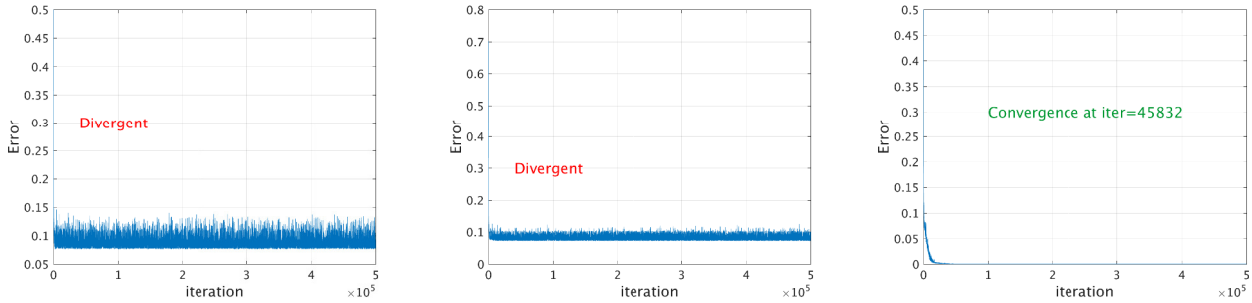


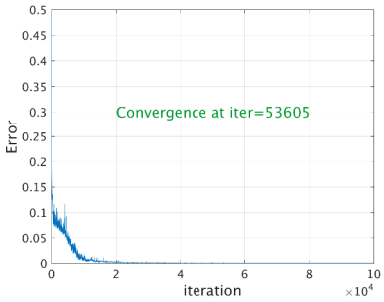


(d) h=10 (e) h=100

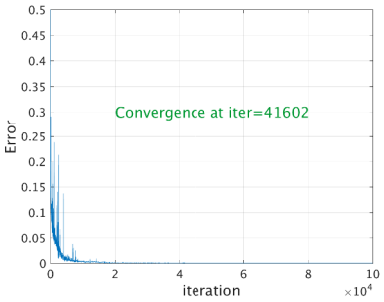
Figure 6: Training errors

(a) h=1 (b) h=2 (c) h=5





(d) h=10



(e) h=100

Part 2: Image classification with CNN features

Stage A: Computing CNN features

Q2A1- Reading an image and then normalizing it to fit the training images size given by `net.normalization.imageSize` is necessary to feed the neural net with the right input. We should also apply any pre-processing the training set has undergone, in this case the substraction of the mean image `net.normalization.averageImage`.

Q2A2- The loaded `net` structure contains 3 fields:

- `layers`: A list of 21 layers, each layer `net.layers{k}` is a structure storing the type of the layer {conv, pool, relu,..} and a reference name {conv%d, pool%d,..}. Each layer depending on its type contains the required implementation parameters: weights for the convolution layers, patch size and pooling method for spatial pooling layers, padding and stride for both...
- `normalization`: Parameters necessary to pre-process the images as seen in Q2A1.
- `classes`: a list of the classification categories, their names and their descriptions.

The output `res` is a list of 22 structures containing each the input of the corresponding layer. The 22th being the final output. it also contains the forward mode computation time and some empty fields, set to store the derivatives of the output w.r to the layer parameters.

Stage B: Image classification using CNN features and linear SVM

Q2B1- We train an SVM linear classifier as seen on *Assignement 2* for each of the three categories {Aeroplane, Motorbike, Person}. The feature vectors are the outputs of the imagenet neural network. First, we only consider the final output `softmax` then we append `fc7` and `fc8` consecutively. We cross-validate the C-regularization parameter of the SVM classifier and report the precision-recall and average precision of the tuned model. We also consider l1 and l2 feature normalization.

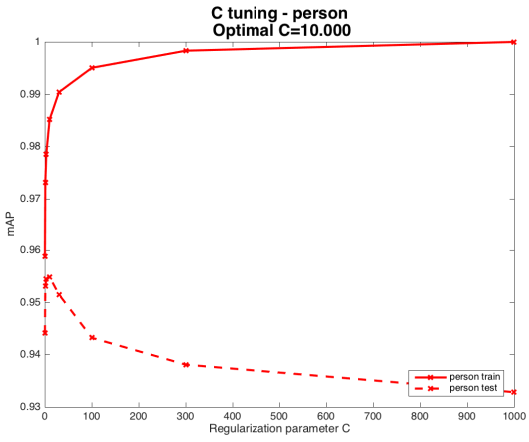
Category	Input	normalization	C	test AP
Motorbike	Softmax	l1	1	82.74%
		l2	.1	86.31%
		∅	1	82.74%
	Softmax + fc8	l1	1000	93.70%
		l2	3	91.32%
		∅	.1	91.15%
	Softmax + fc8 + fc7	l1	1000	93.46%
		l2	1	90.34%
		∅	.1	90.33%
Aeroplane	Softmax	l1	.1	91.97%
		l2	.1	91.78%
		∅	.1	91.97%
	Softmax + fc8	l1	1000	93.72%
		l2	3	92.15%
		∅	3	90.82%
	Softmax + fc8 + fc7	l1	1000	93.51%
		l2	3	91.94%
		∅	.1	91.82%
Person	Softmax	l1	3	83.18%
		l2	.1	84.28%
		∅	3	83.18%
	Softmax + fc8	l1	1000	95.46%
		l2	10	93.29%
		∅	3	93.23%
	Softmax + fc8 + fc7	l1	1000	95.47%
		l2	3	94.69%
		∅	3	94.64%

Table 3: SVM with neural net outputs - statistics

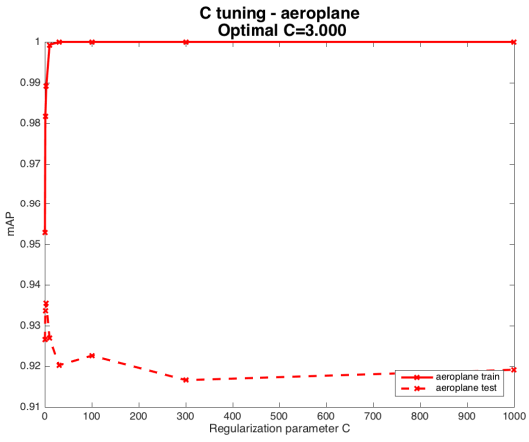
Feature normalization improves the model's performane up to 3 points, which is significant. The cross validation is also important, especially that for a given class, different features or different normalization are optimally tuned with extremely distinct C-parameter (figure 6) ranging from .1 to 1000. A default C wouldn't perform very well in all settings.

Figure 6: C-tuning

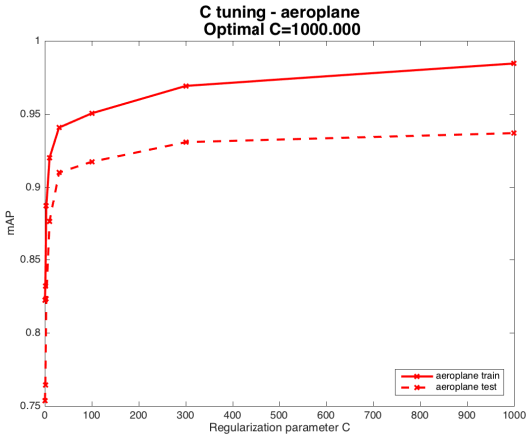
(a) person, l2 , features= softmax + f8



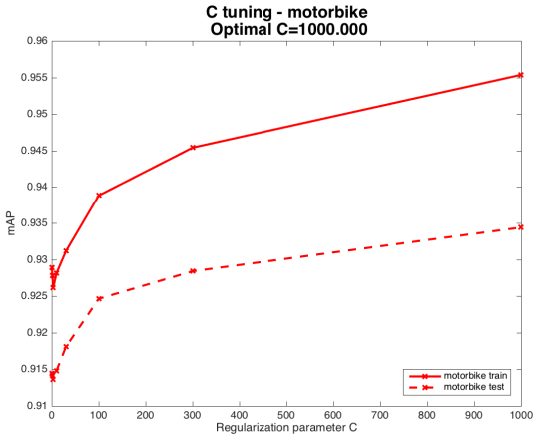
(b) person, l2 , features= softmax + f8 + fc7



(c) aeroplane, l1 , features= softmax + f8



(d) motorbike, l1 , features= softmax + f8 + fc7



although the imagenet classification problem doesn't consider a *person* category, the SVM on the neural network output seems to perform very well in all considered settings. For norm=l2 and features=softmax+fc8+fc7 we gete 36/36 correctly retrieved in the top 36 test images (figure 7). To explain this performance we list in table (4) the output classes of the neural networks for the top 10 ranked images in the SVM classifier:

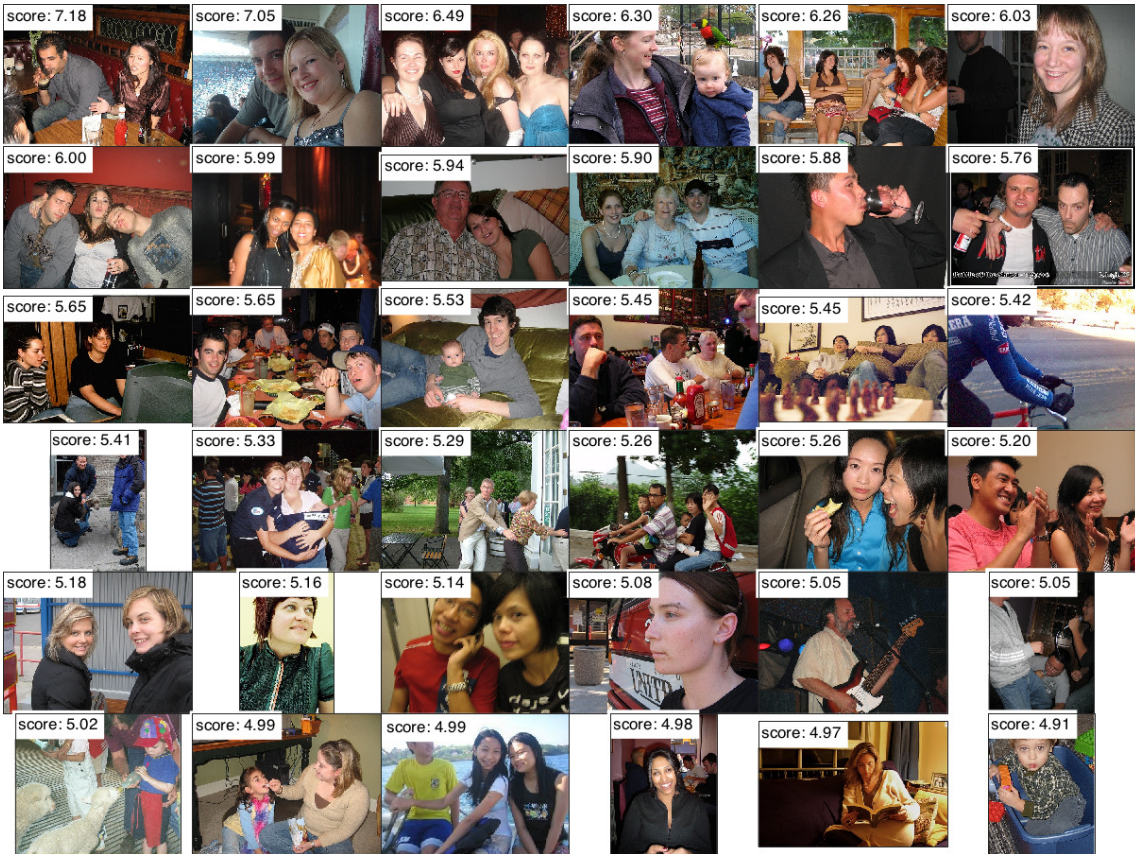


Figure 7: Top 36 - person catgeory - norm=l2 - features=softmax+fc7+fc8

Class 1	Class 2
groom, bridegroom	bassoon
seat belt, seatbelt	sunscreen, sunblock, sun blocker
limousine, limo	miniskirt, mini
ice lolly, lolly, lollipop, popsicle	carousel, carrousel, merry-go-round, roundabout, whirligig
miniskirt, mini	volleyball
suit, suit of clothes	crutch
feather boa, boa	pajama, pyjama, pj's, jammies
torch	theater curtain, theatre curtain
fur coat	pajama, pyjama, pj's, jammies
carousel, carrousel, merry-go-round, roundabout, whirligig	groom, bridegroom

Table 4: Neural network - top 2 probable classes for the top 10 SVM scored images

Hence the net is implicitly trained to capture human presence through other classes such as {groom,suit,skirt,pyjama...} where $\mathbb{P}(person|class = 1)$ is high.

Q2B2 From *Assignement 2*, the top performances in every setting are reported in (table 5)

Table 5: AP on test sets: {VLAD, BoVW, FV} vs NNet

		aeroplane	motorbike	person
Linear	BoVW	0.55	0.48	0.71
	VLAD	**0.75**	0.69	0.76
	FV	0.70	**0.73**	**0.77**
	NNet (l1,{softmax+fc8+fc7})	**0.93**	**0.93**	**0.95**

It's obvious from the results above that the neural network encoding outperforms the VLAD, BoVW and FV of *Assignement 2* i.e that deep learning the features is more efficient that hand-designing them. Nonetheless, the model fails to classify some images shown below. Generally there are more FP than FN (Person: FN=149, FP=85, Motorbike: FN=29, FP=1, Aeroplane: FN=43, FP=2), we can presume that the co-existence of multiple classes in one image blurs the SVM output. We'll have to admit though that the misclassified images aren't trivial.

Figure 8: Aeroplane

(a) False positives (exhaustive)

(b) False negatives

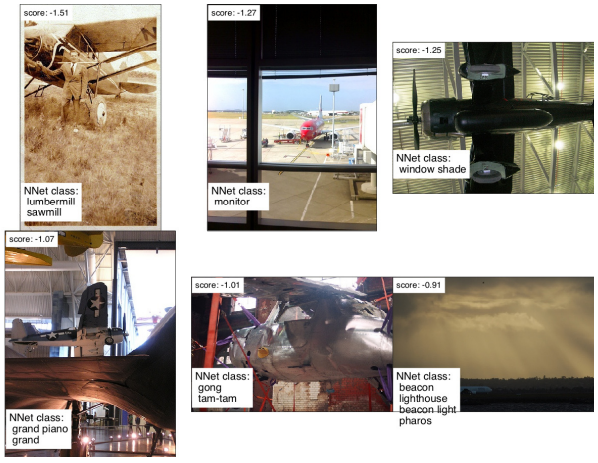


Figure 9: Motorbike

(a) False positives (exhaustive)

(b) False negatives

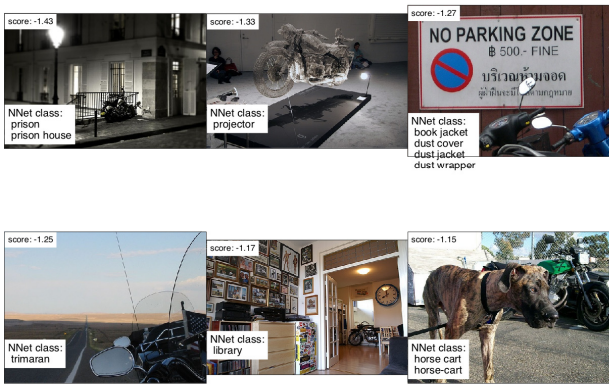
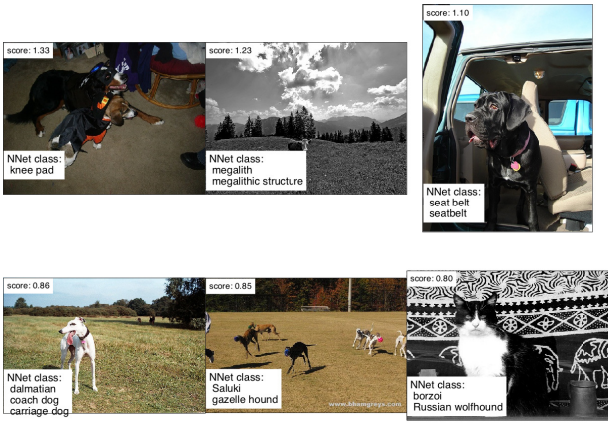


Figure 10: Person

(a) False positives



(b) False negatives