Type recognition of PDEs using Deep Learning

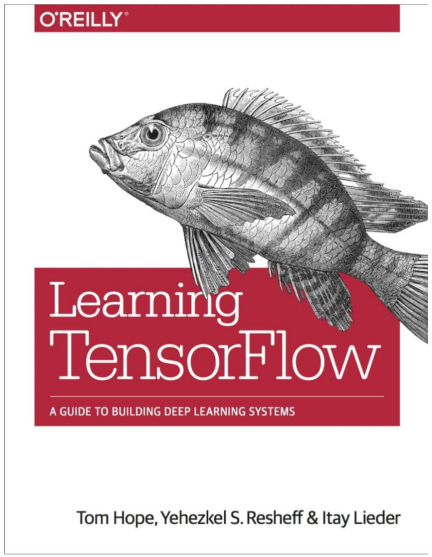
Olivier Pironneau[1]

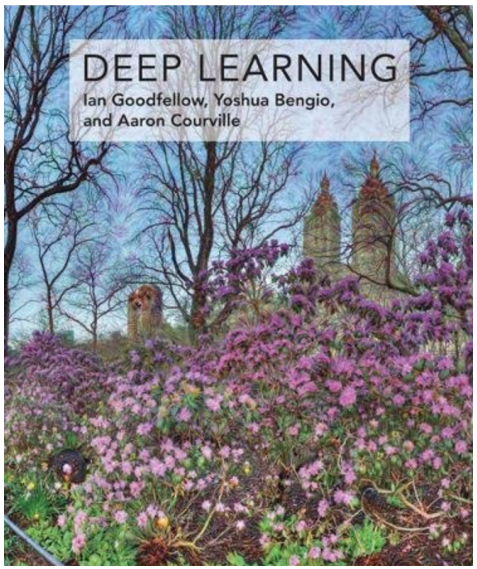
[1]University of Paris VI (UPMC), Laboratoire J.-L. Lions (LJLL)
http://ann.jussieu.fr/pironneau

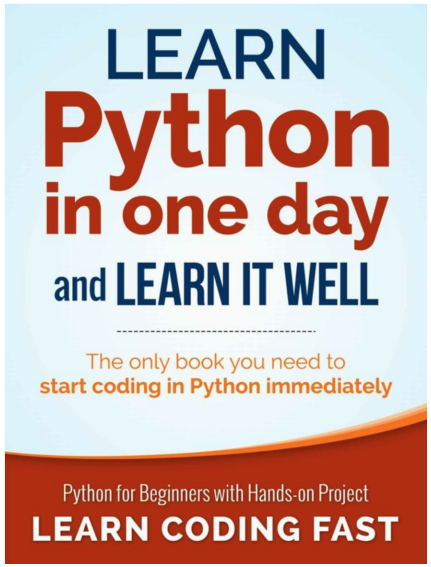
Acknowledgement: this exercise follows a discussion with Georges Oppenheimer and Michel Bercovier

Content

1. Classification of PDE
2. Numerical method
3. Identification Using TensorFlow (deep learning)







Classification of PDE

$u(\cdot, t)$ is x, y-periodic in $(0,1) \times (0,1)$ and

$$\partial_t u - c_1 \partial_{xx} u - c_2 \partial_{yy} u = 0, \quad u(x,0) = u_0(x).$$
or
$$\partial_{tt} u - c_1 \partial_{xx} u - c_2 \partial_{yy} u = 0, \quad u(x,0) = u_0(x), \partial_t u(x,0) = 0.$$

where c_1 or c_2 are random in $(0,1)$.
We combine both with c_0 randmly 0 or 1:

$$c_0 \partial_t u + (1 - c_0) \partial_t tu - c_1 \partial_{xx} u - c_2 \partial_{yy} u = 0,$$

which is randomly the heat or the wave equation in $Q = (0,1)^2 \times (0,T)$

$$\bar{u}(x,y,T) = \frac{u(x,y,T) - \min_Q u}{\max_Q u - \min_Q u} \text{ is observed}$$

To make it more difficult we observe only a noised u in a small sub-square

$$w(x,y) = \omega(x,y)\bar{u}(x,y,T), \quad \forall x, y \in (d,1) \times (d,1)$$

where $1 - \mathit{noiselevel} \leq \omega(x,y) \leq 1$ is random and $d \in (0,1)$.

Numerical Method

- For an easy implementation we have used a space-time Lagrangian finite element method of degree 2 on a uniform grid.
- This is similar to a centered implicit (similar to Crank-Nicolson in time) finite difference.
- For stability the viscous term is always greater than ϵ. Hence the following is solved:

Find $u \in V_h$ with $u(x,0) = u_0(x)$, such that for all $u_h \in V_h$

$$\int_{(0,1)\times(0,T)} [-(1-c_0)\partial_t u \partial_t \hat{u} + c_0 \partial_t u \hat{u} + c_1 \partial_x u \partial_x \hat{u} + c_2 \partial_y u \partial_y \hat{u}] = 0$$

with $c_0 = 0$ or 1 and $c_i \in (\epsilon, 1$ random.

Implementation with freefem++

```
real[int] cc(3);  // random coefficients

fespace Vh(Th,P2, periodic=[[1,x,z],[3,x,z],[4,y,z],[2,y,z]]);
Vh  u,uh;

problem pde(u,uh) = int3d(Th)(  // the PDE
                              (  (1-cc[0])*dz(u) )*uh - cc[0]*dz(u)*dz(uh) // z is time
                             + (0.01+0.1*cc[1])*dx(u)*dx(uh) + (0.01+0.1*cc[2])*dy(u)*dy(uh)
                             ) + on(5, u= sin(pi*x)+sin(2*pi*y));  // initial conditions
```

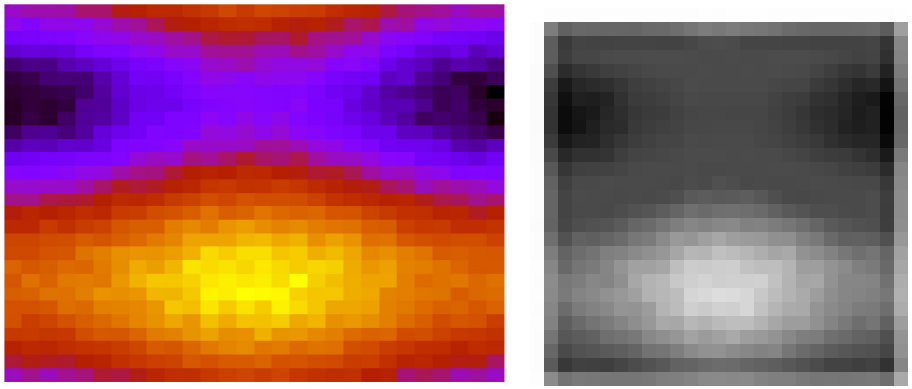
Generate the training set

```
for(j=0;j<nclass;j++){
      for(int k=1;k<Ntrain;k++){
        ofstream fff("ffdata/training-images/"+j+"/train"+k+".txt");
        cc[0]=j;
        cc[1]=randreal1();
        cc[2]=randreal1();
        cout<<j<<" j, cc "<<cc[1]<<"  "<<cc[2]<<endl;
        pde;
        real umax=u[].max, umin=u[].min ;
        if(umax-umin<1e-10) umax=umin+1e-10;  // data will be rescale to be in (0,1)
        for(int k=0; k<th.nv; k++)
            w[][k]=(1+noise*randreal1())/(1+noise)
                * (u(th(k).x,th(k).y,T)-umin)/(umax-umin);  // blur output with noise  and scale to (0,1)
        plot(w, fill=1);
        for(int jj=0;jj<=jjmax;jj++){
            for(int ii=0;ii<=iimax;ii++)
                fff << w(1-box+ii*box/iimax,1-box+jj*box/jjmax) << " "; // Take from a window (1-box,1)x(1-box,
            fff<<endl;
        }
        fff.flush;
        if(withjpg) pgnuplot << "set output 'ffdata/training-images/"+j+"/train"+k
            +".jpg'\n plot 'ffdata/training-images/"+j+"/train"+k
            +".txt'  matrix with image \n set output \n" ;
  }
}
```

Generate the test set

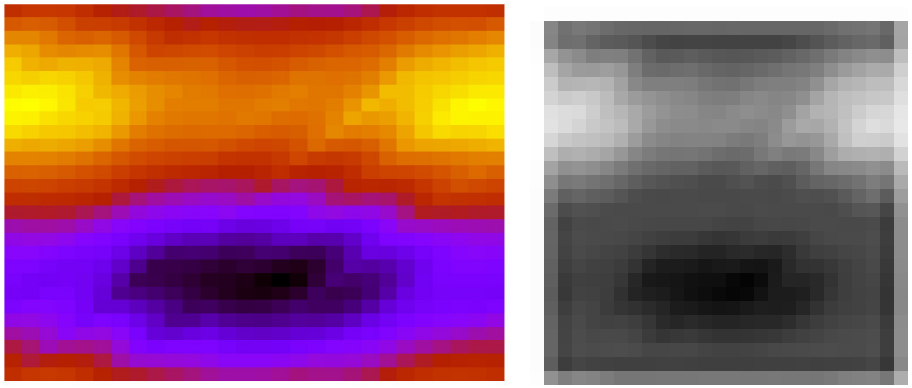
```
// test images
for(int k=1;k<Ntest;k++){
    cc[0] = ( randreal1()  > 0.5);
    j = cc[0];
    ofstream fff("ffdata/test-images/"+j+"/test"+k+".txt");
    cc[1]=randreal1();
    cc[2]=randreal1();
    cout<<j<<" j, cc "<<cc[1]<<"  "<<cc[2]<<endl;
    pde;
    real umax=u[].max, umin=u[].min ;
    if(umax-umin<0.01) umax=umin+0.01;
    w=u(x,y,T);
    plot(w, fill=1);
    for(int jj=0;jj<=jjmax;jj++){
        for(int ii=0;ii<=iimax;ii++)
            rfff << xu = noise*randreal1()
                        +(u(1-box+ii*box/iimax,1-box+jj*box/jjmax,T)
                            -umin)/(umax-umin)/(1+noise) <<" ";
        fff<<endl;
    }
    fff.flush;
    if(withjpg) pgnuplot << "set output 'ffdata/test-images/"+j+"/test"+k
        +".jpg'\n plot 'ffdata/test-images/"+j+"/test"+k+".txt'  matrix with image \n set output \
}
```

Gnuplot 3D Space-Time Results



case 0

$$: \partial_t u - c_1 \partial_{xx} u - -c_2 \partial_{yy} u = 0.$$

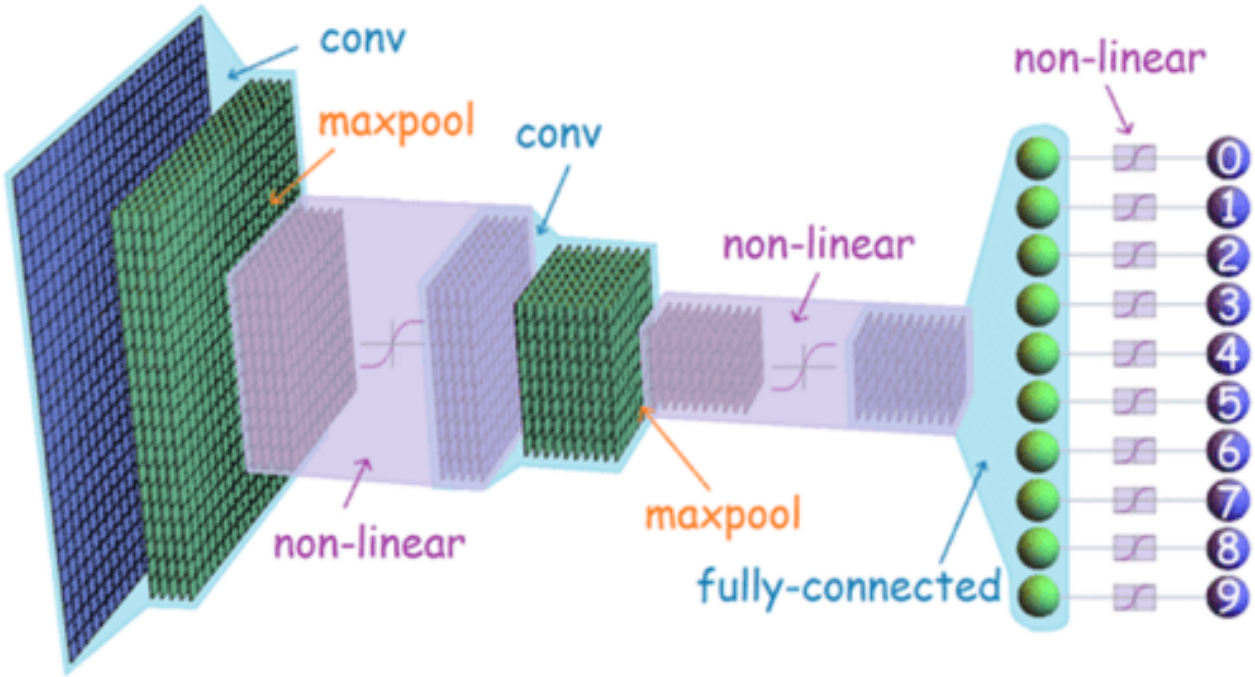


case 1

$$: \partial_{tt} u - c_1 \partial_{xx} u - -c_2 \partial_{yy} u = 0$$

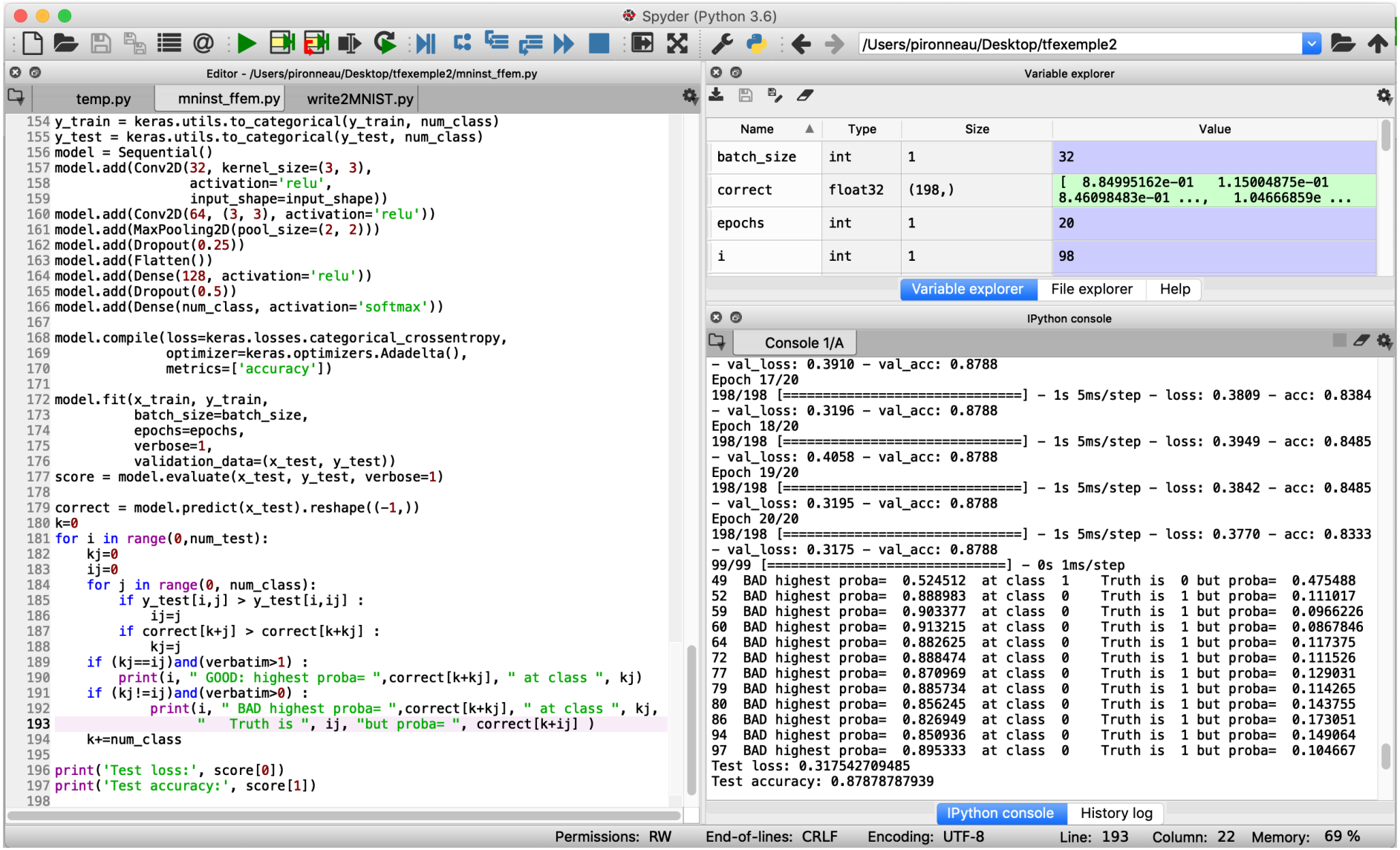
Training and testing

1. There are 2x100 training images into folder labelled by $j = 0, 1$ and
2. 100 randomly generated test images into folders labelled by their type.
3. The neural network is the same as the MNIST test case in Keras
4. Accuracy is the percentage of images whose label has been identified.

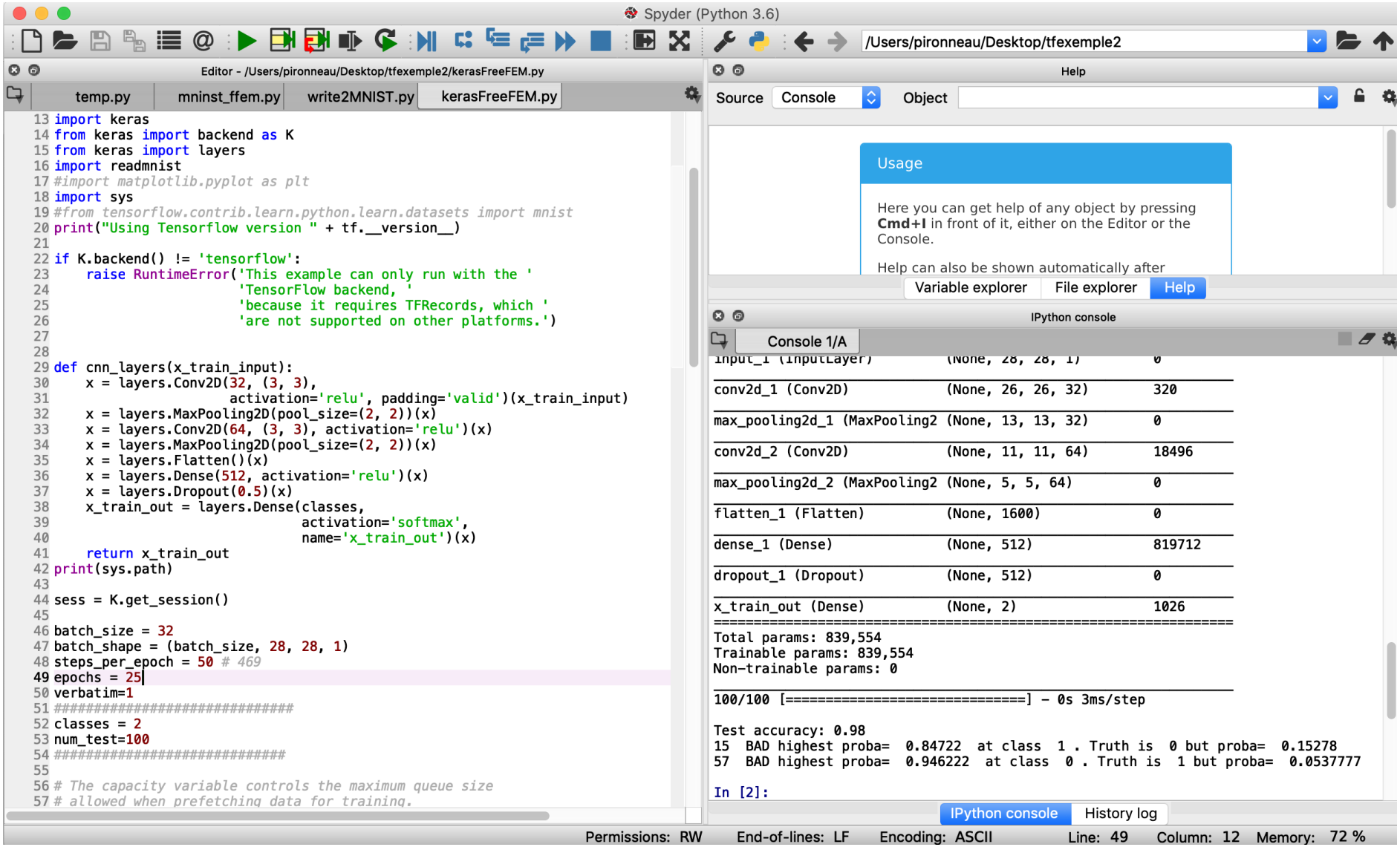


This is a case with 10 classes

Tensorflow+Keras in Anaconda+spyder => Precision 0.87



Tensorflow using 28x28 png images => Precision 0.98



Conclusion

- Perfect recognition is achieved even with a small level of noise
- Even by observing a small portion of the final time solution
- But performance decay if noise is too high and/or window too small
- The most important conclusion is that freefem++ is a good data generated to test and learn AI

To Install these software:

- install `anaconda` with `python 3.7`
- On Mac it is useful but not essential to install `homebrew`
- install `keras` from within Anaconda
- In case of an nVidia GPU update to `keras-gpu` (Windows and Ubuntu only)