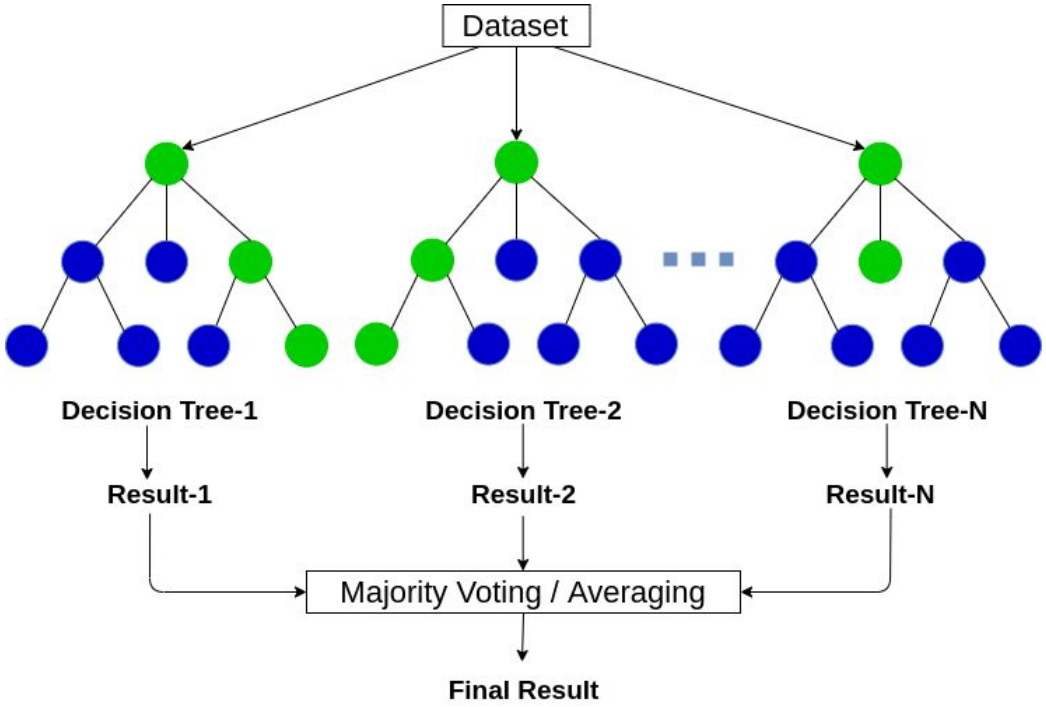
Report from #project-group-11

Shenli Tang, Chang Sun, Junhao Yin, Atsuyuki Yamada

Tree based algorithm: Random Forest and Adaboost

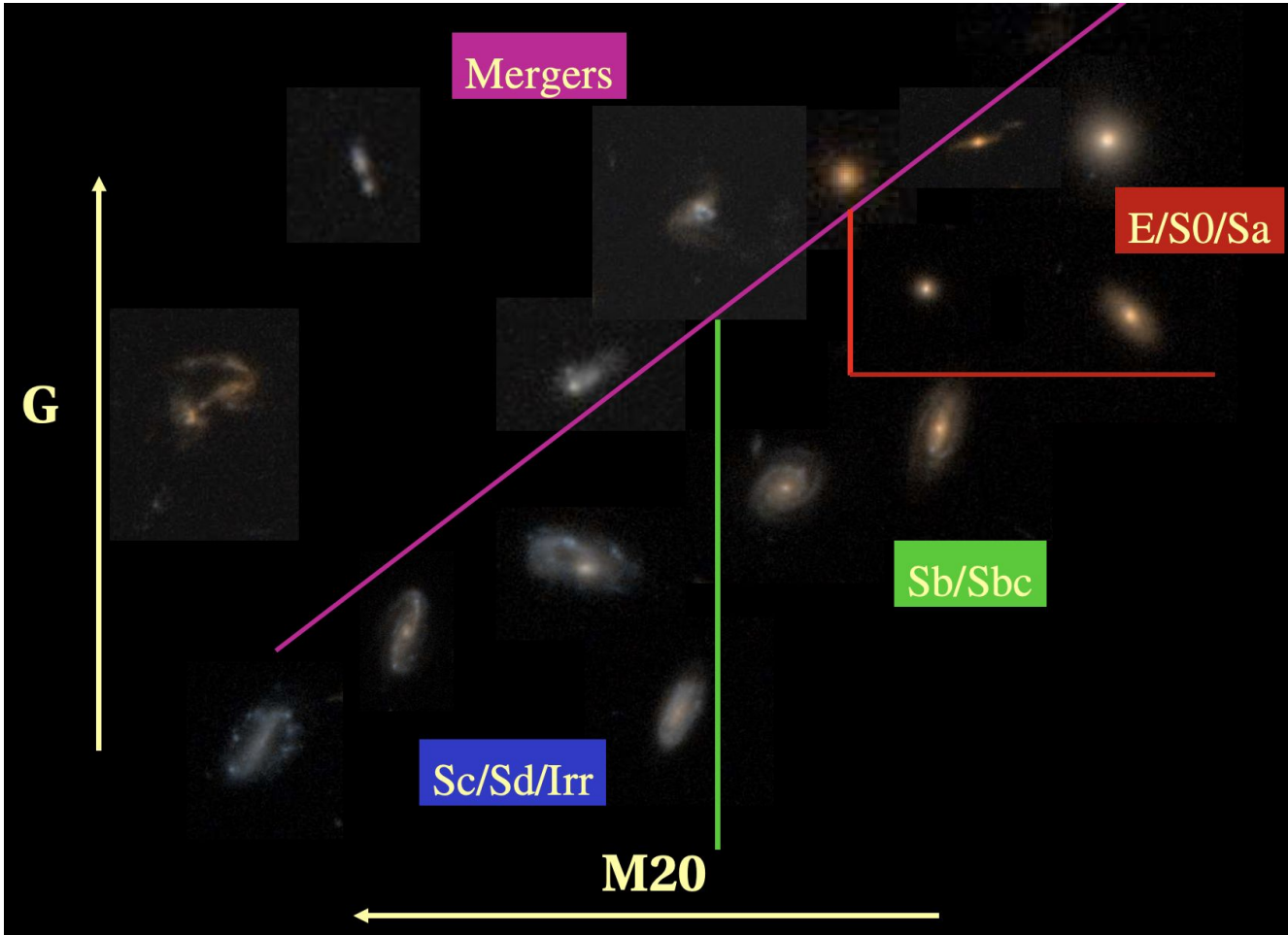
Is an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.



The basic idea of random forest is **bagging** (bootstrap aggregating). Works for low bias and high variance data. On the other hand, **Adaboost** works for high bias and low variance data, and it is very sensitive to the outliers in the data set.

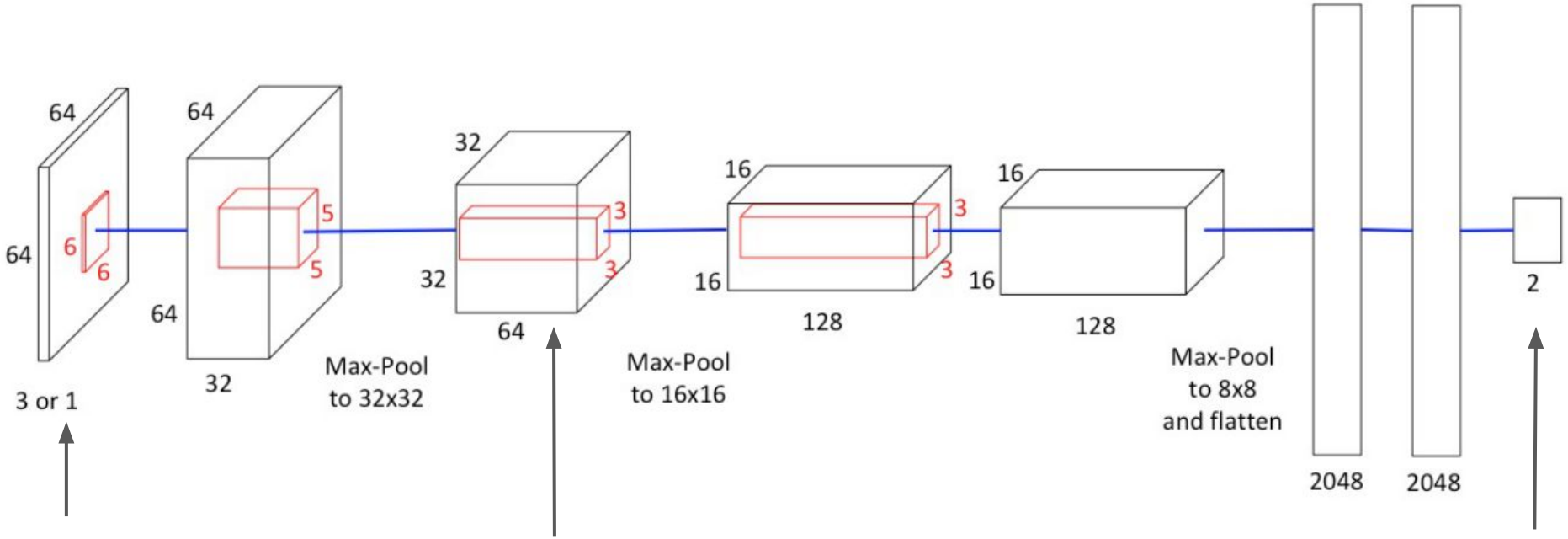
Random Forest for galaxy classification

Galaxy classification: G and M20 factors can be directly measured from the image of the galaxies. The boundaries of different types of galaxies have been studied. Random forest will help us to classify these galaxy types when new image data are achieved.



CNN for galaxy classification

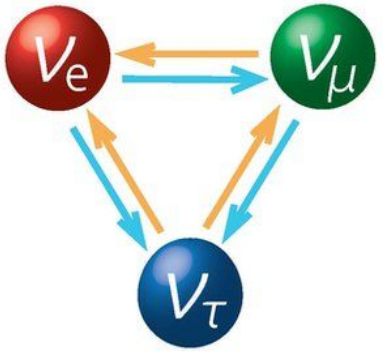
CNN is an algorithm based on MLP, it linearize the problems. CNNs work as a **feature extraction machine**, and is seen as a **geometric data transformer**.



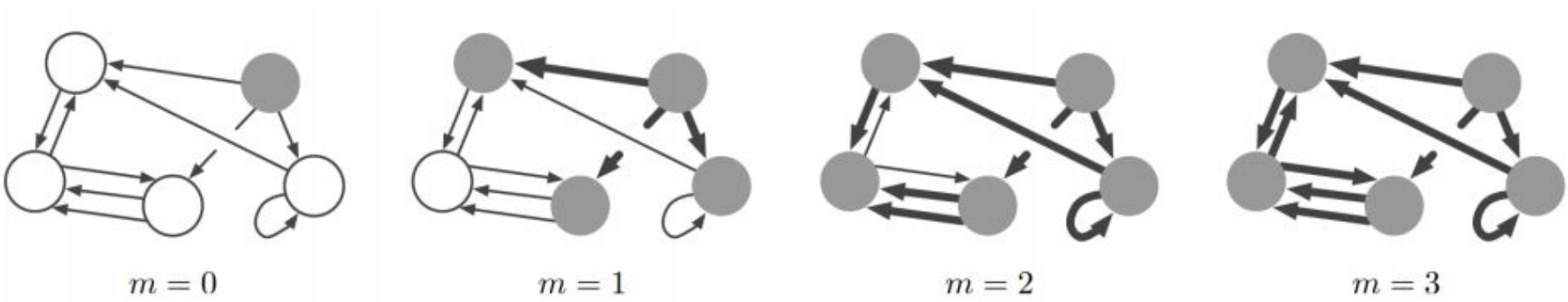
GNN for Neutrino oscillation

Different from CNN, which study on the information locally, GNNs **transfer messages globally**.

GNNs are not good choice for classifying galaxies, because we expect the interaction between different components of a galaxy should be local, rather than global.



Describe the fraction of different flavors of Neutrinos after travelling time t in the space.



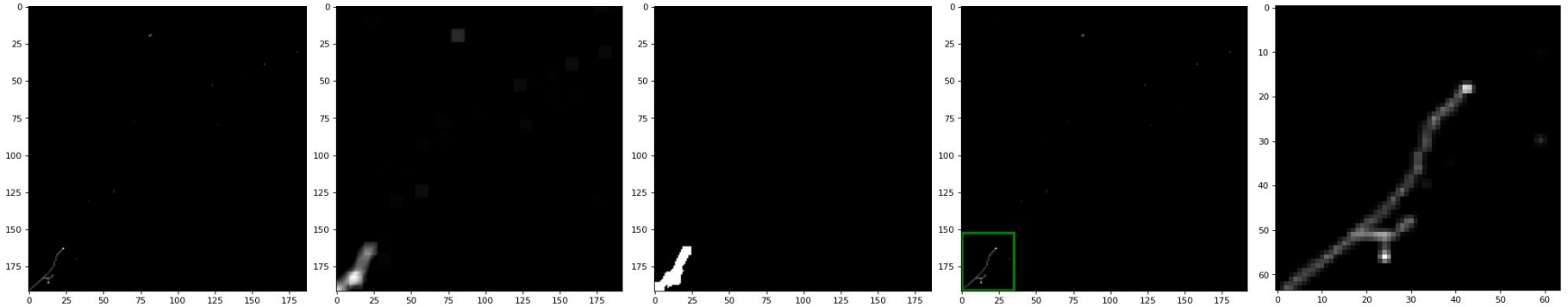
Project results: NN for particle classification from Sun Chang

Preprocessing:

1. Apply 9*9 average blurring
2. Apply Threshold of 0.16
3. Crop out rectangle enclosing all non-zero pixels, with edge shift = +10 pixels
4. Bilinear resize cropped image into 64*64, float32 image

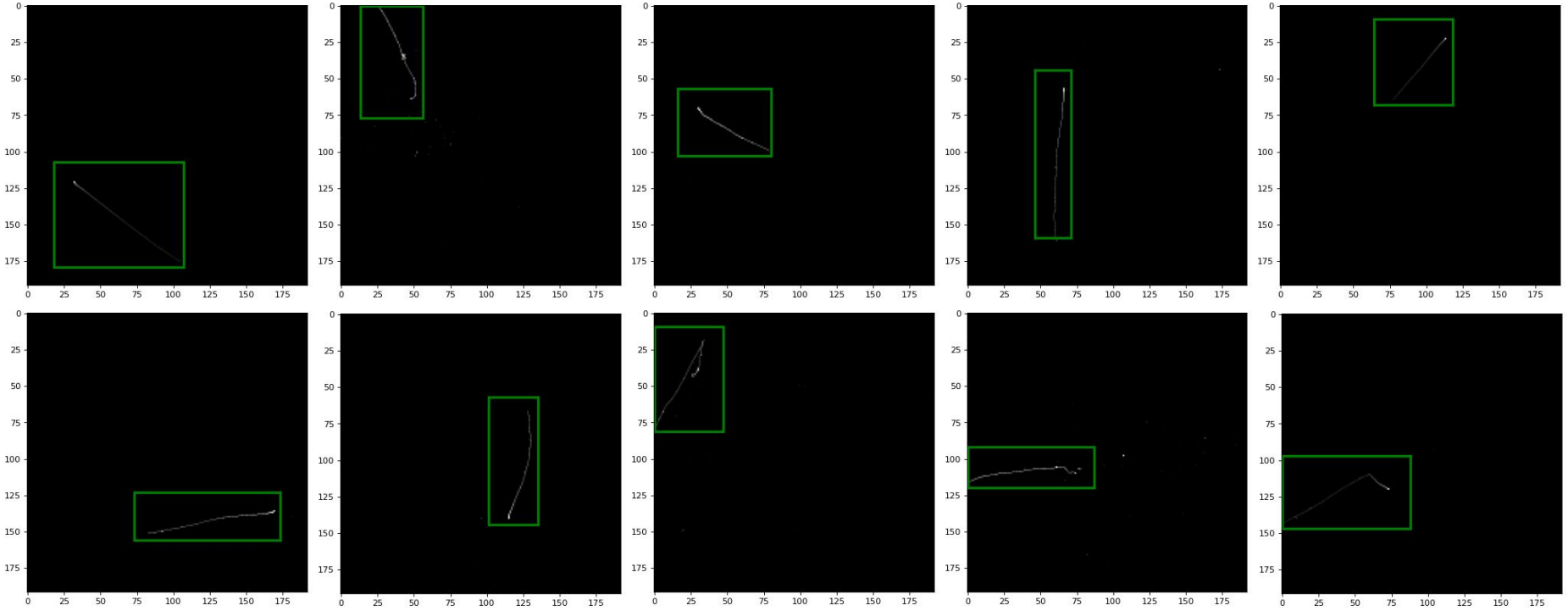
```
def crop(img):
    thresh = cv.threshold(cv.blur(img,(9,9)),0.06,1,cv.THRESH_BINARY)[1].astype(dtype=np.uint8)
    x,y,w,h = cv.boundingRect(cv.findNonZero(thresh))

    y= y-10 if y > 9 else 0
    yy= y+h+10 if y+h <181 else 191
    x= x-10 if x > 9 else 0
    xx= x+w+10 if x+w <181 else 191
    return cv.resize(img[y:yy,x:xx],(64,64))
```

Network Structure is as shown on the right.



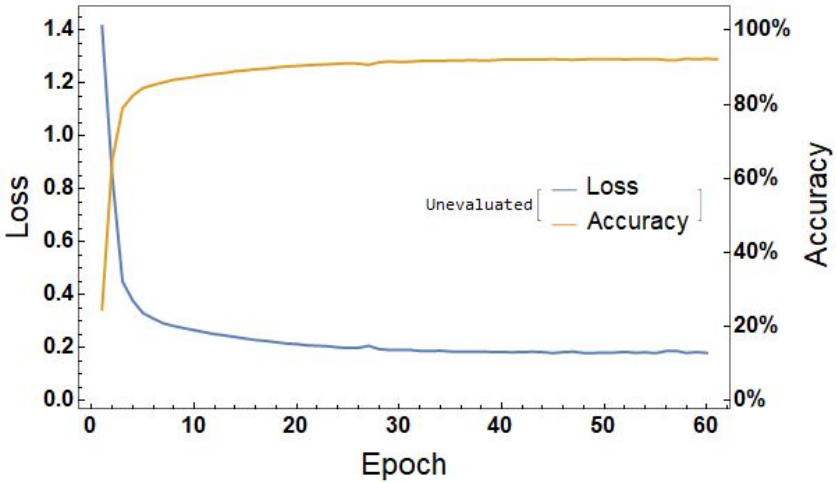
Example Workflow, Training Set, ID=3. Avg. cost: 0.31ms*cpu

Some Example Preproced Images



My Network

Trained with batch_size=2.5k, Adam(lr=1e-4).
Constructed by tensorflow keras
Best test_accuracy=92.35%, with test_loss=0.189 at epoch 57.
Performence: ~0.544ms / 1000 samples on GTX1060 6g
Computation cost: ~10^0 M flop

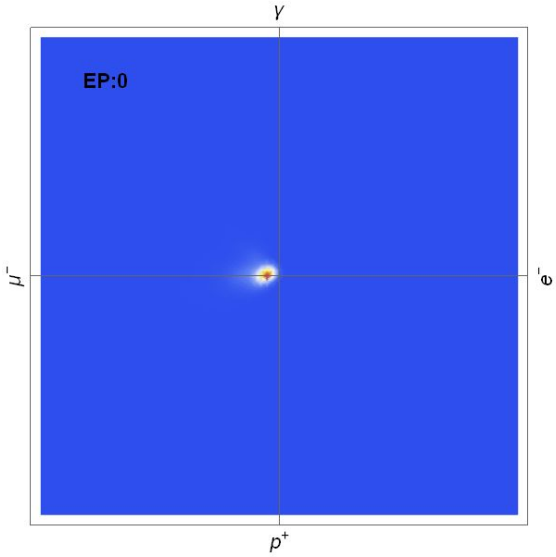


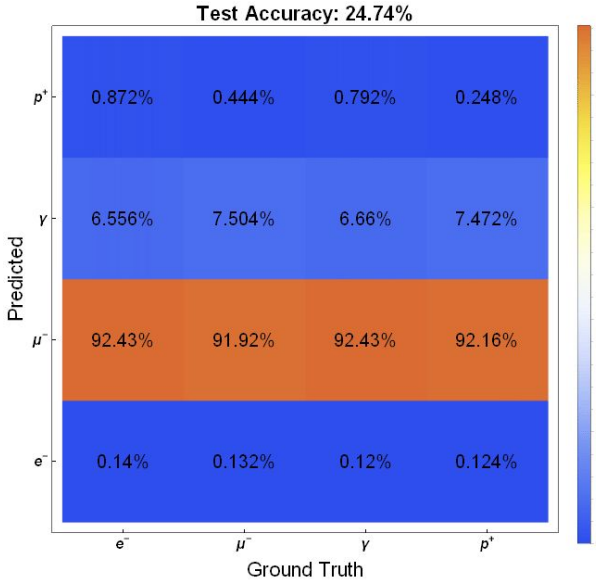
Model: "Mini-PID"

Layer	Options	Output Shape	Param #
Reshape		(None, 64, 64, 1)	0
Conv2D	[k=5,s=2,LReLU(0.3)]	(None, 32, 32, 48)	1248
Conv2D	[k=5,s=2,LReLU(0.3)]	(None, 16, 16, 48)	57648
Conv2D	[k=3,s=1,LReLU(0.3)]	(None, 16, 16, 96)	41568
MaxPool2D	[size=2]	(None, 4, 4, 96)	0
Flatten		(None, 1536)	0
Dense	[ReLU]	(None, 192)	295104
Dropout	[0.25]	(None, 192)	0
Dense	[ReLU]	(None, 128)	24704
Dropout	[0.25]	(None, 128)	0
Dense	[ReLU]	(None, 16)	2064
Dense	[Softmax]	(None, 4)	68

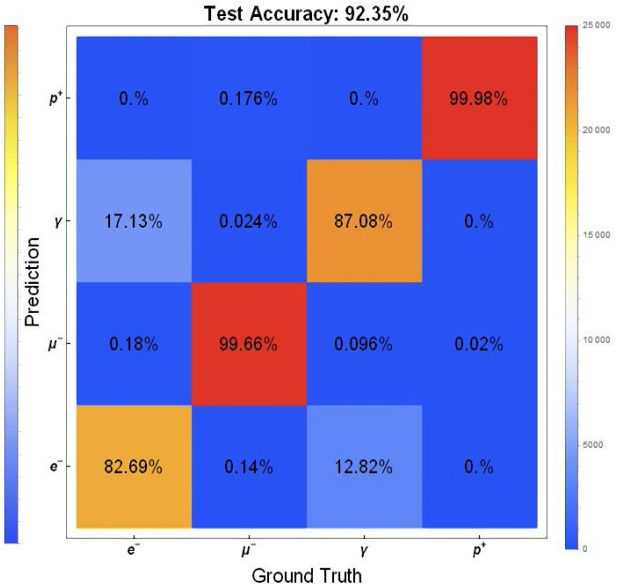
Total params: 422,404
Trainable params: 422,404
Non-trainable params: 0

Visualized Training Process





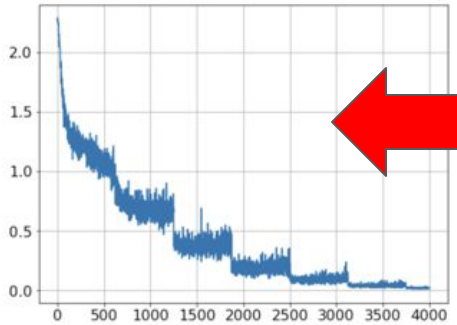
EP57: Best Test Accuracy



Challenge A from Junhao Yin

1. Understand data
2. make a model
   a. Just copy from the lecture
3. make a training module
   a. Also copy and paste… modify to adapt data here
4. train!

batch[64]	
data	Tensor[192,192]
pdg	Tensor[1]
label	Tensor[1]
index	Tensor[1]




```
def train_torch(data_loader, model, num_iterations=1, lr=0.001, optimizer='SGD', gpu=False):
    # Create a Binary-Cross-Entropy (BCE) loss module
    criterion = torch.nn.CrossEntropyLoss()
    # Create an optimizer
    optimizer = getattr(torch.optim,optimizer)(model.parameters(),lr=lr)
    # Now we run the training!
    loss_v=[]
    while num_iterations > 0:
        for data in data_loader:

            #   if gpu:
            #       data,label = data.cuda(),label.cuda()
            # Prediction
            prediction = model(data['data'])
            # Compute loss
            loss = criterion(prediction, data['label'])
            # Update weights
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            # Record loss
            loss_v.append(loss.item())
            # Brake if we consumed all iteration counts
            num_iterations -= 1
            if num_iterations < 1:
                break

    return np.array(loss_v)
```

```
class MLP(torch.nn.Module):
    def __init__(self, num_filters=16):

        super(MLP, self).__init__()
        # MLP w/ 2 hidden layers, 128 neurons each
        self._classifier = torch.nn.Sequential(
            torch.nn.Linear(192*192, num_filters),
            torch.nn.LeakyReLU(),
            torch.nn.Linear(num_filters,10)
        )

    def forward(self, x):
        # Make 2d image into 1D array
        x_1d = x.view(-1, np.prod(x.size()[1:]))
        return self._classifier(x_1d)
```