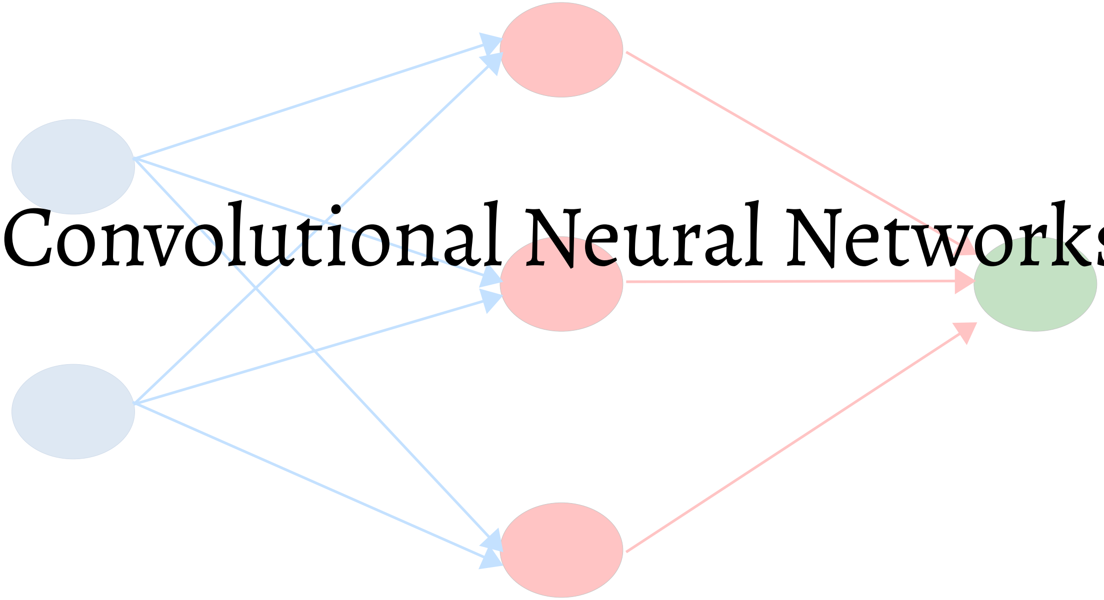
# PH6232: Machine Learning for Physics applications



## Convolutional Neural Networks

Mar 10th, 2023

# CNNs

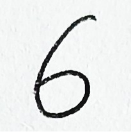
When inputs have specific relations to each other, or when input information has structural inter-relations which *matter* ....

For example, images.

# CNNs

When inputs have specific relations to each other, or when input information has structural inter-relations which *matter* ....

Say I was identifying handwritten numerals…

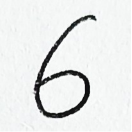




Inputs could be pixel by pixel intensity of the image… is a given pixel black or not.
This is a 200x200 pixel image, so total 40000 numbers.

# CNNs

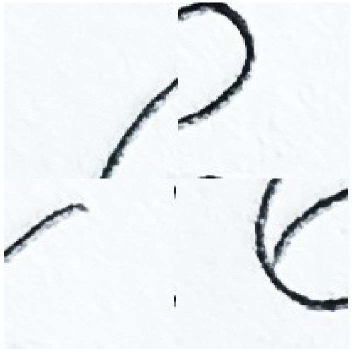
When inputs have specific relations to each other, or when input information has structural inter-relations which *matter* ....

Say I was identifying handwritten numerals…



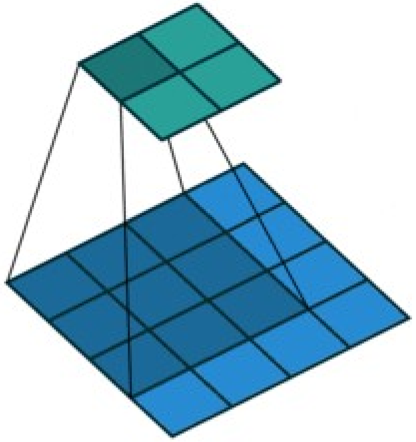
But the order of these 40000 numbers carries information!

*not same as*



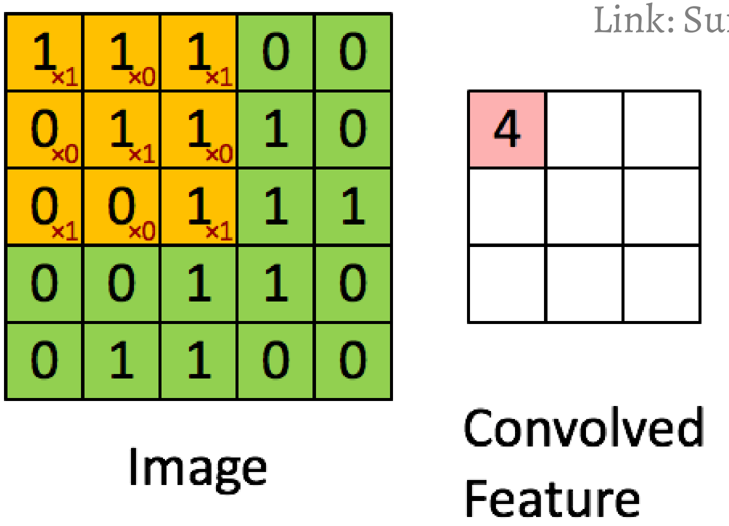
*Moreover, are all 40k numbers really needed?*

# Enter convolutions…



arXiv:1603.07285 Vincent Dumoulin, Francesco Visin

Link: Sumit Saha

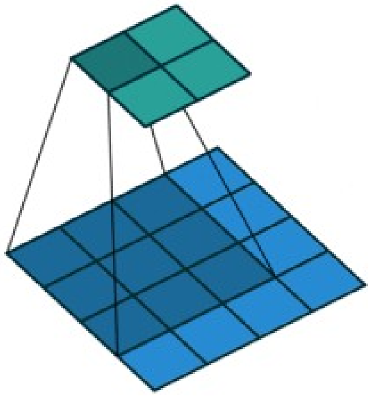



Here, the filter (or kernel) is run over the image moving from left to right, top to bottom

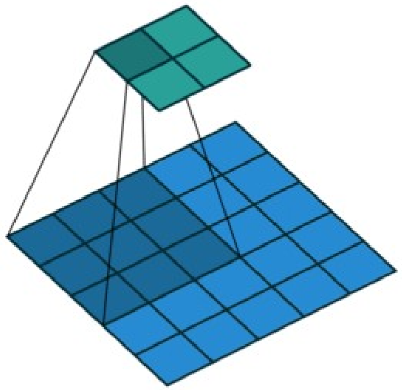
Filter is $\begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix}$ , performing a Hadamard product.

# Terminology

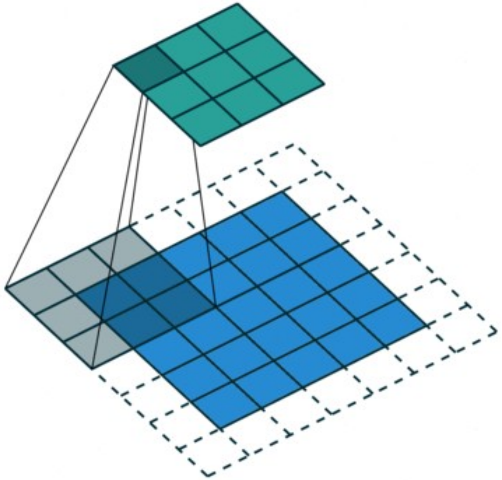
arXiv:1603.07285 Vincent
Dumoulin, Francesco Visin



3x3 filter, stride=1



3x3 filter, stride=2

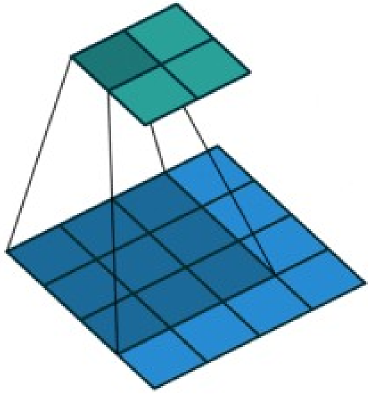


3x3 filter, stride=2, padded

Stride is step (in x or y) between successive operations

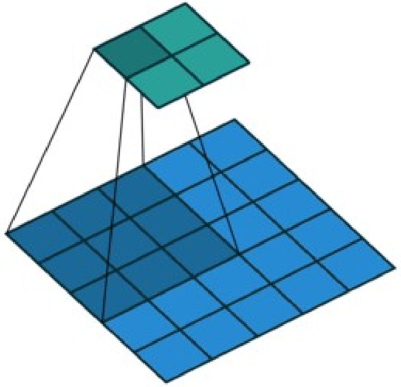
# Terminology

arXiv:1603.07285 Vincent Dumoulin, Francesco Visin



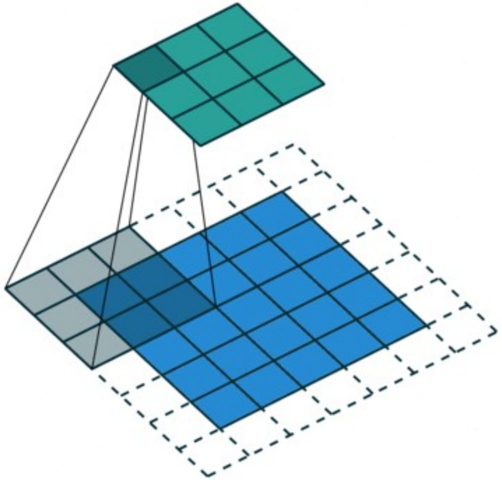
3x3 filter, stride=1

4x4 → 2x2



3x3 filter, stride=2

5x5 → 2x2



3x3 filter, stride=2, padded

5x5 is padded to 7x7

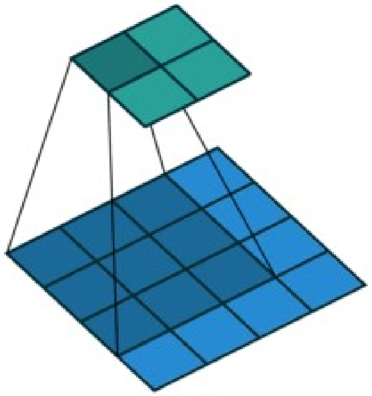
5x5 → 3x3

Input nxn, output is mxm

m = (width-kernel)/stride + 1   (round up)

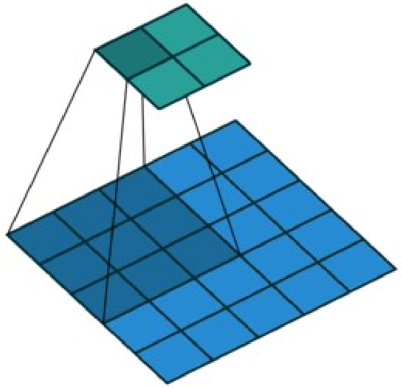
# Terminology

arXiv:1603.07285 Vincent Dumoulin, Francesco Visin



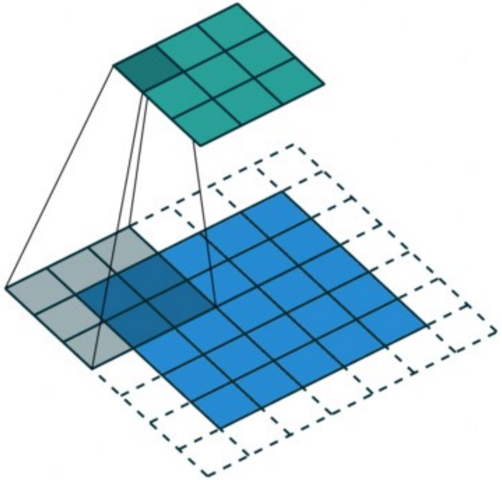
3x3 filter, stride=1

4x4 → 2x2



3x3 filter, stride=2

5x5 → 2x2



3x3 filter, stride=2, padded

5x5 is padded to 7x7

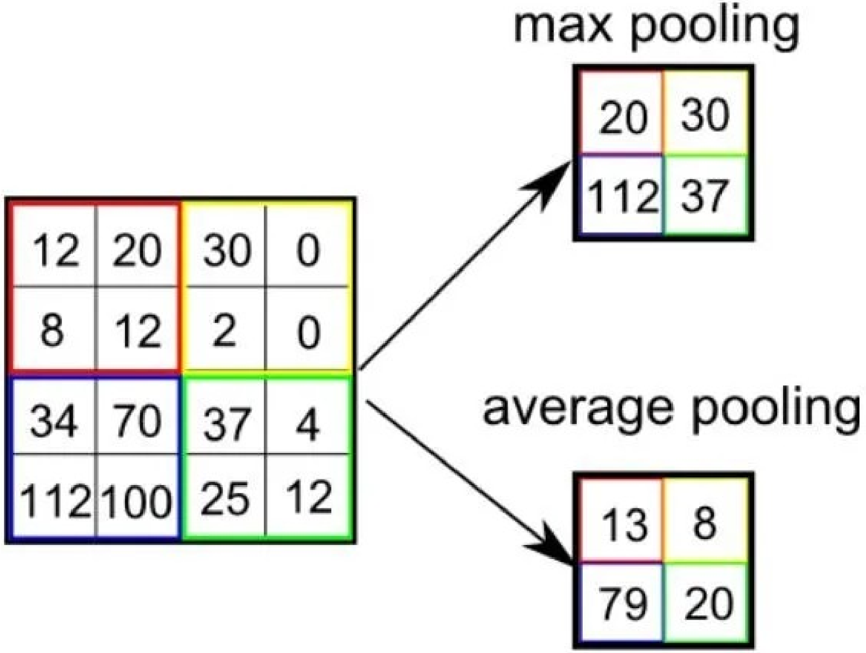
5x5 → 3x3

Input nxn, output is mxm

m = (width-kernel)/stride + 1   (round up)

Padding ensures edge of image gets read in enough too.

# MaxPooling

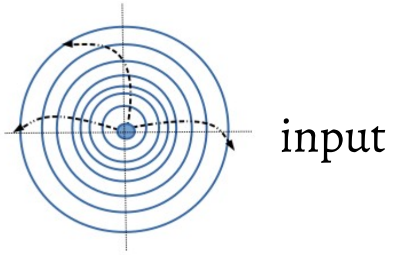


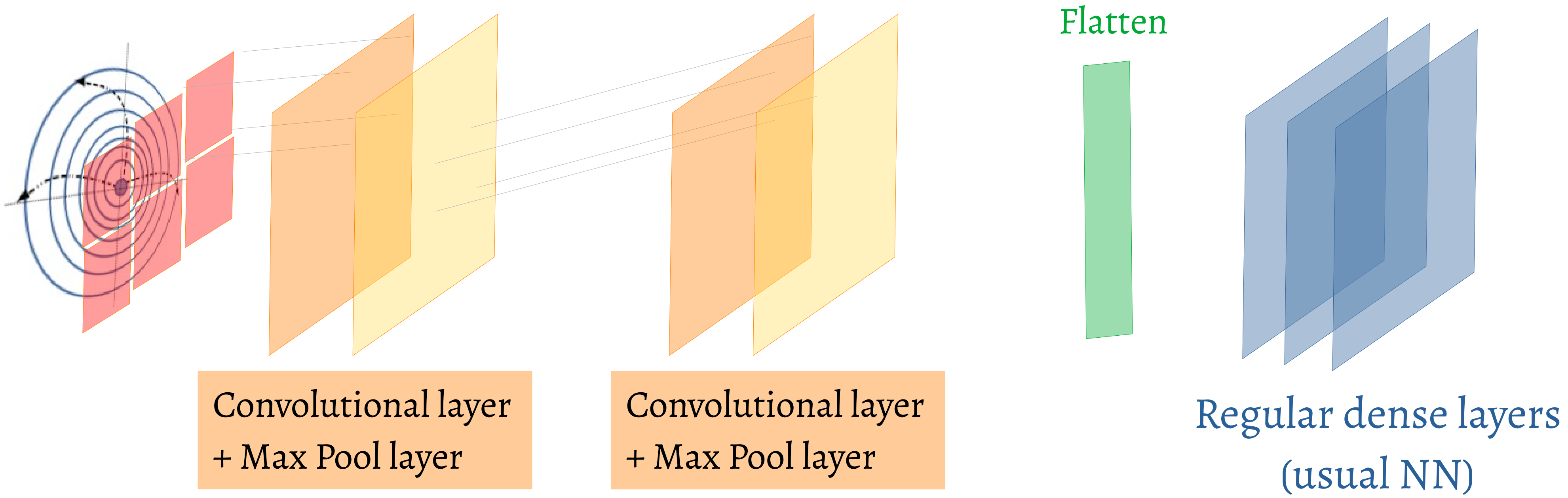

For 2x2 maxpooling, take the max value of a 2x2 block, and stride across

For 2x2 averagepooling, take the average value of 2x2 block, and stride across

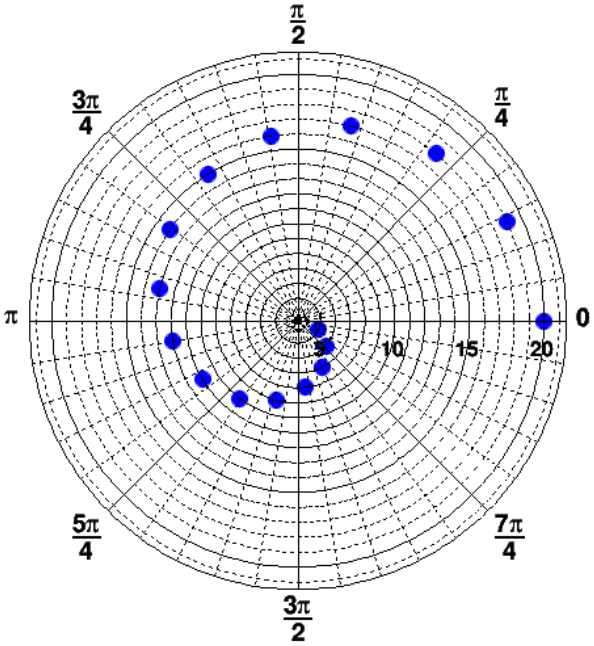
Link: Sumit Saha

# Putting it together



# Let’s take an example



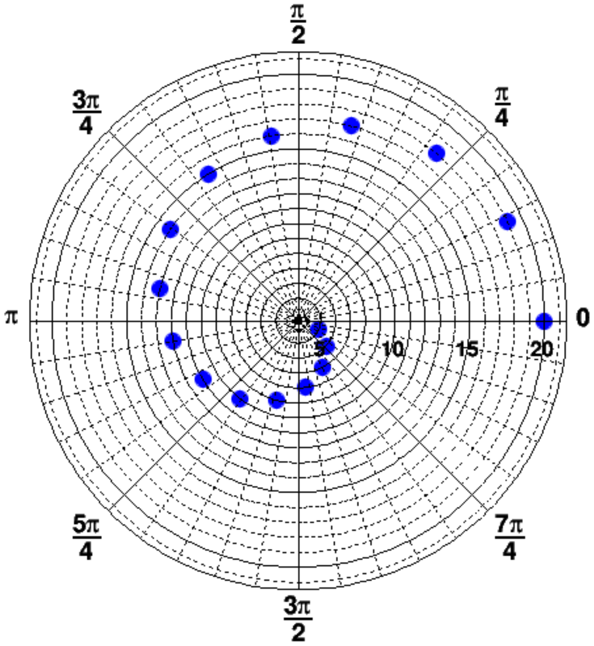

Here we have 16 detectors, yielding 16 points.

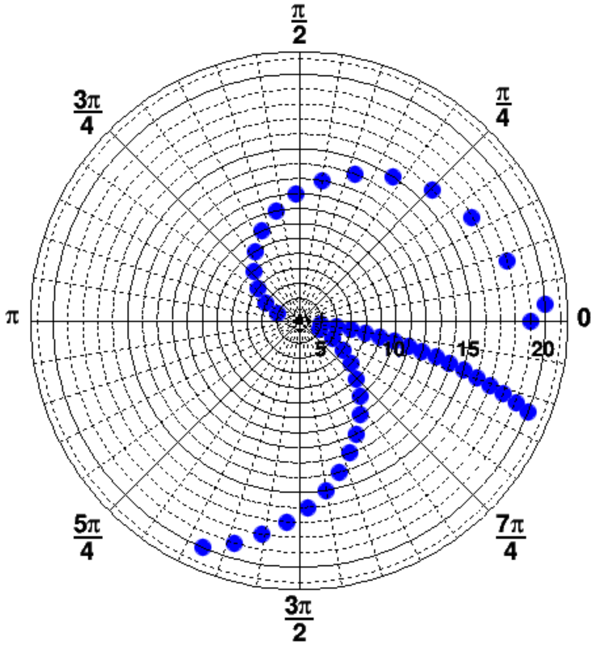
Charged particles bend in a magnetic field. The magnitude of their momentum decides the radius of curvature.

We measure the “points” where the particle intersects our detector. A “fit” to the points will give the trajectory of the particle.

Figure shows one charged particle trajectory in a “spectrometer” designed to measure momentum.

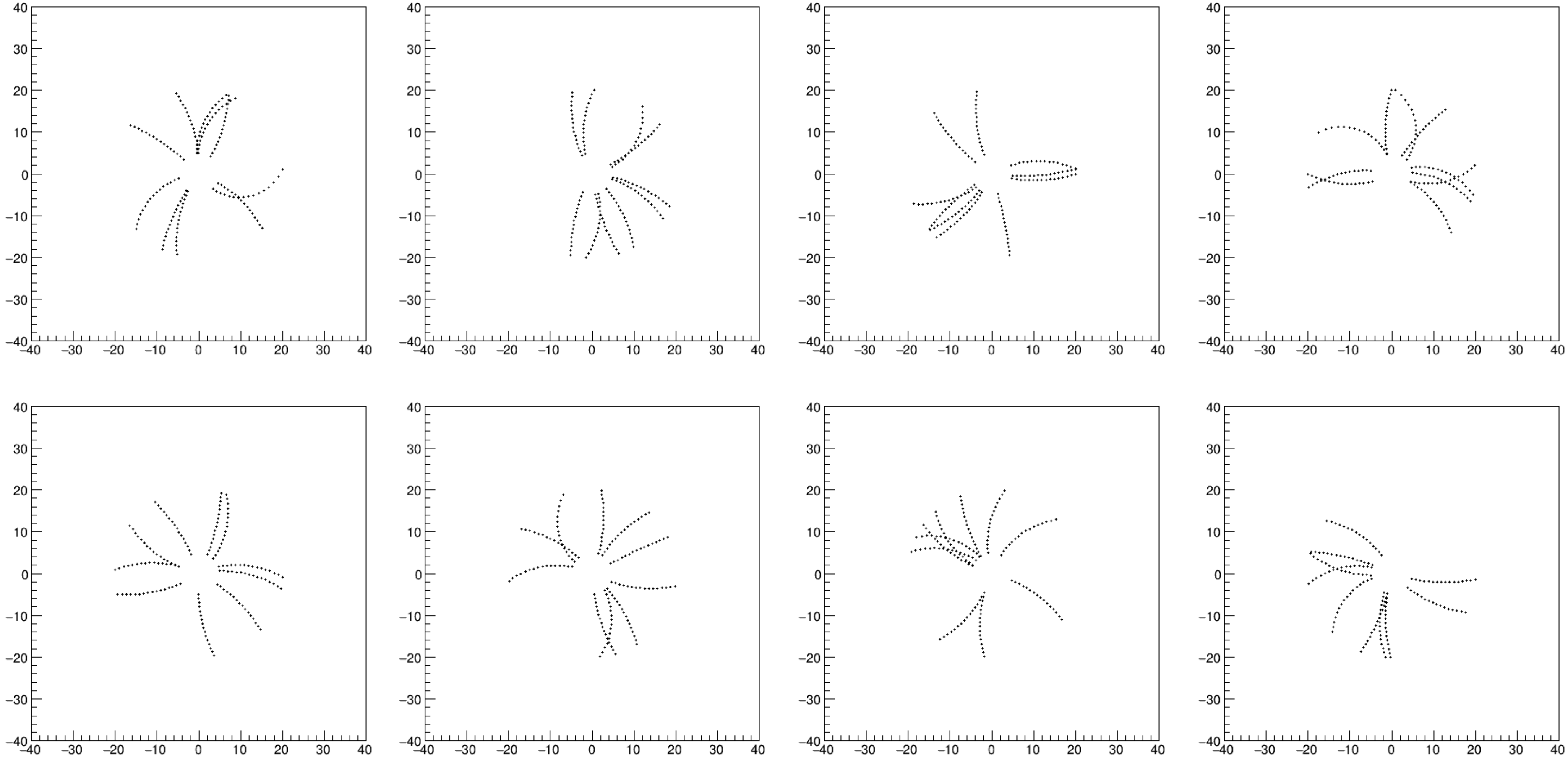
# Let’s take an example




Right figure shows three charged particle trajectories. One of these is “straighter” indicating a larger momentum (large radius of curvature = large momentum)

Each image has 10 charged particles, one of which may be of high momentum. Can you tell which of these contain a “high momentum” particle?



# We setup a CNN

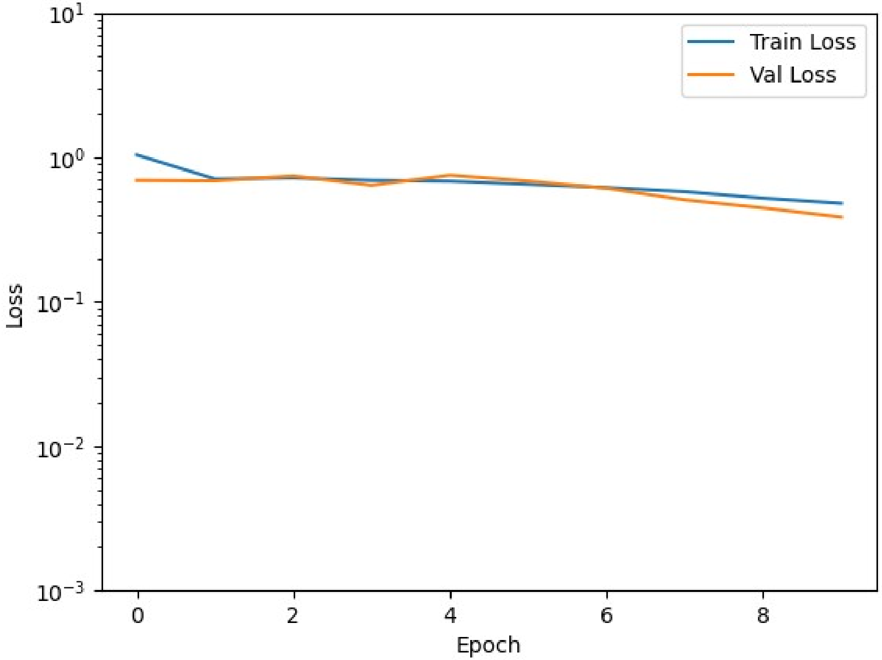
```
model = Sequential()
model.add(Conv2D(32, (5,5), strides=(1,1),
activation='relu',kernel_initializer='he_uniform', padding='SAME',
input_shape=(IMG_HEIGHT,IMG_WIDTH,1)))
model.add(MaxPool2D((2,2), padding='SAME'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3,3), strides=(1,1),
activation='relu',kernel_initializer='he_uniform', padding='SAME'))
model.add(MaxPool2D((2,2), padding='SAME'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(64, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(32, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, epochs=10, verbose=0, validation_data=test_data,
callbacks=cb)
```

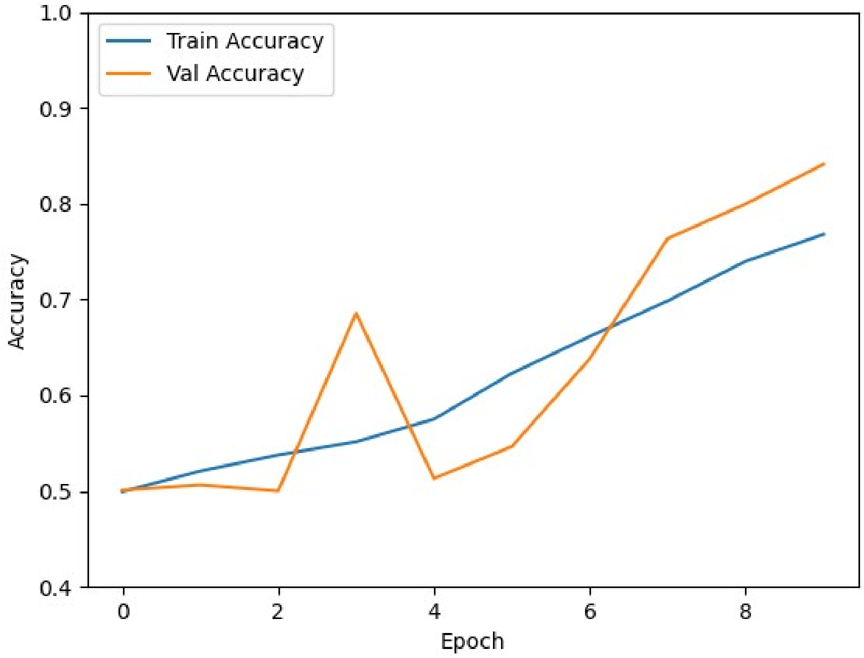
# Runtime is slow...

1) Resize images from 512x512 to 128x128 to save computation
2) Save model for best val_accuracy while training

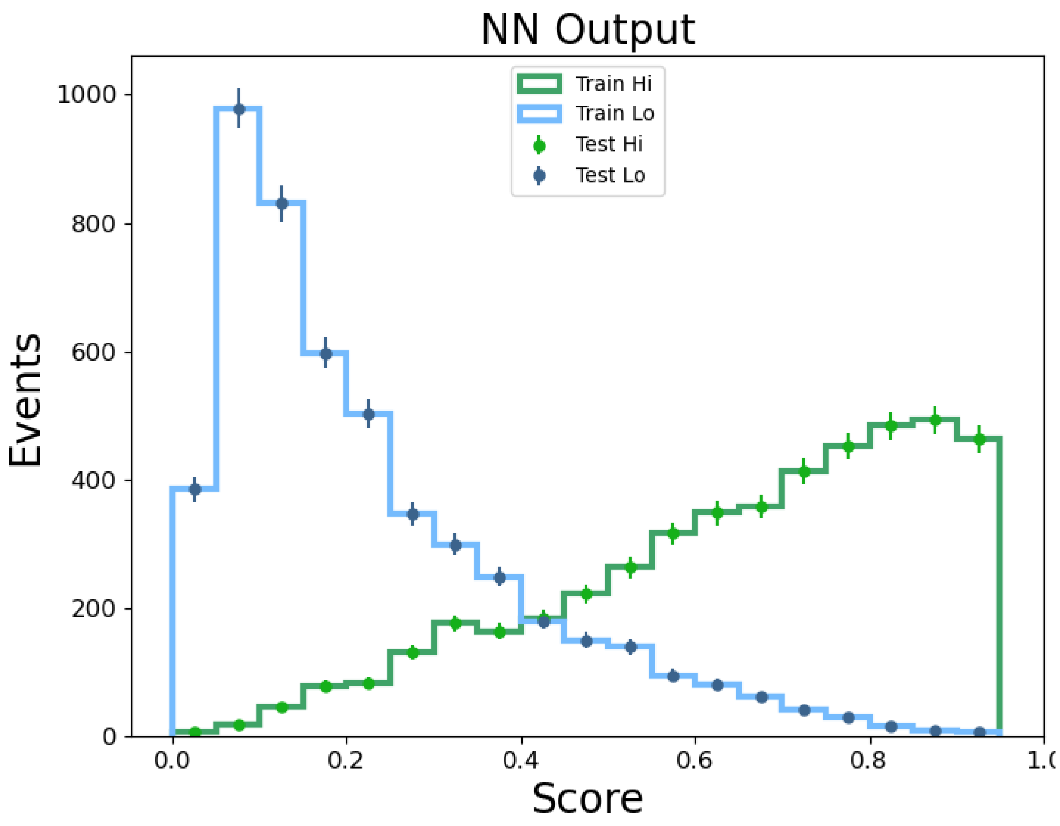
```
Epoch 1: val_accuracy improved from -inf to 0.50100, saving model to best_model.h5
Epoch 2: val_accuracy improved from 0.50100 to 0.50640, saving model to best_model.h5
Epoch 3: val_accuracy did not improve from 0.50640
Epoch 4: val_accuracy improved from 0.50640 to 0.68570, saving model to best_model.h5
Epoch 5: val_accuracy did not improve from 0.68570
Epoch 6: val_accuracy did not improve from 0.68570
Epoch 7: val_accuracy did not improve from 0.68570
Epoch 8: val_accuracy improved from 0.68570 to 0.76360, saving model to best_model.h5
Epoch 9: val_accuracy improved from 0.76360 to 0.79970, saving model to best_model.h5
Epoch 10: val_accuracy improved from 0.79970 to 0.84110, saving model to best_model.h5
```

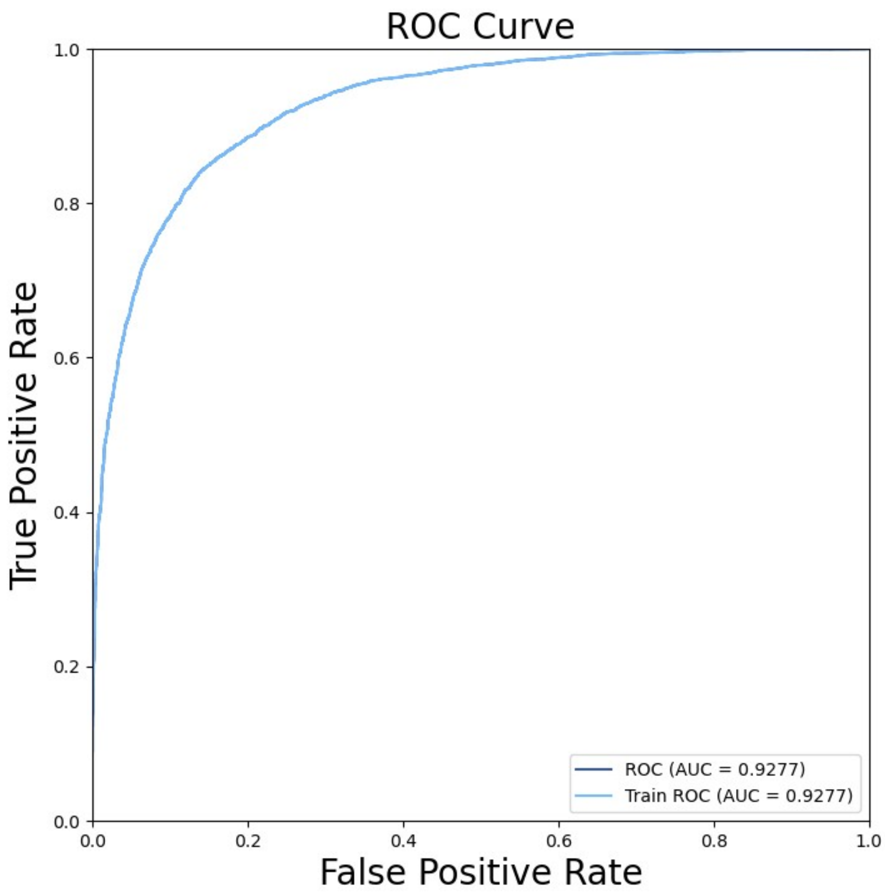
# Results


Train Loss
Val Loss
$10^{1}$
$10^{0}$
$10^{-1}$
$10^{-2}$
$10^{-3}$
Loss
0
2
4
6
8
Epoch


Train Accuracy
Val Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Accuracy
0
2
4
6
8
Epoch

# Results


NN Output
Train Hi
Train Lo
Test Hi
Test Lo
Events
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
Score


ROC Curve
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
ROC (AUC = 0.9277)
Train ROC (AUC = 0.9277)

# Code etc.

Not expecting you to implement this code.

But, the code (as a python file) and the input data is linked from the course webpage.

If you feel like playing with the CNN, go ahead. Just remember, its pretty resource-intensive.