# Counting corners in polygons. Dataset 1

**Samuele Sottile**

Lund University

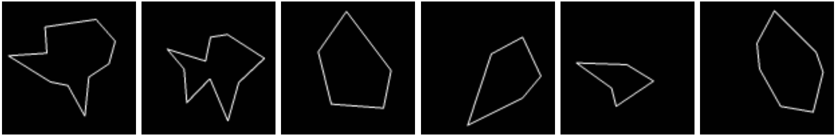
Introduction to deep learning – October 25, 2021

# Outline

# Description of the dataset

- Images of poligons with 3 to 10 sides.
- Images have 100x100x1 size. 100x100 pixel and 1 is represents the colour of the pixel (white or black)
- Training dataset: 5000 images. Test dataset: 5000 images.

Example of dataset



## Pre-procession of data

- Import library: from tensorflow.keras.utils import to_categorical
- trg = to_categorical(trg[0:n]-3)
- One hot encoding: to_categorical converts integers into binary class matrix: it prepares the labels for the categorical cross-entropy. The labels are $d_1, d_2, \cdots, d_8$, where the label is $d_1 = (1,0,0,0,0,0,0)$, $d_2 = (1,0,0,0,0,0,0)$, $\cdots$, $d_8 = (0,0,0,0,0,0,1)$.

# Choice of the model

- Since we are dealing with images we use a **Convolutional Neural Network** (CNN).
- **Two convolutional layers:** one to detect lines and one to detect corners.
- The **kernel** of the convolutional layers must have size comparable to the thickness of lines in terms of pixels (Guess: 5x5).
- **Zero padding** in order not to decrease the size of the image after the convolution.
- Small number of **hidden layers** in the MLP and non linear activation function in the hidden layers (RELU)
- **Output activation function:** softmax. It gives to probability of being in each class.
- Number of **output nodes** equal the numbers of classes: 8.
- **Loss function:** categorical cross entropy since we are dealing with classification problems.
- **Optimizer:** ADAM to keep the past gradient on the count for a faster optimization process.

# CNN architecture

```
# Define the CNN model
model = Sequential([
    Conv2D(60, kernel_size=(6, 6), padding='same', activation='relu',
           input_shape=input_shape),
    MaxPooling2D(pool_size=(5, 5)),
    Conv2D(60, kernel_size=(5, 5), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(4, 4)),
    Flatten(),
    Dropout(0.3),
    Dense(15, activation='relu'),
    Dense(8),
    Activation('softmax')
])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 100, 100, 60)      2220
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 20, 20, 60)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 20, 20, 60)        90060
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 60)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 1500)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 1500)              0
_________________________________________________________________
dense_8 (Dense)              (None, 15)                22515
_________________________________________________________________
dense_9 (Dense)              (None, 8)                 128
_________________________________________________________________
activation_4 (Activation)    (None, 8)                 0
=================================================================
Total params: 114,923
Trainable params: 114,923
Non-trainable params: 0
```

# MLP architecture

Hyperparameters:

- Learning rate = 0.005
- Dropout = 0.3
- Epochs = 30
- Batch size = 80

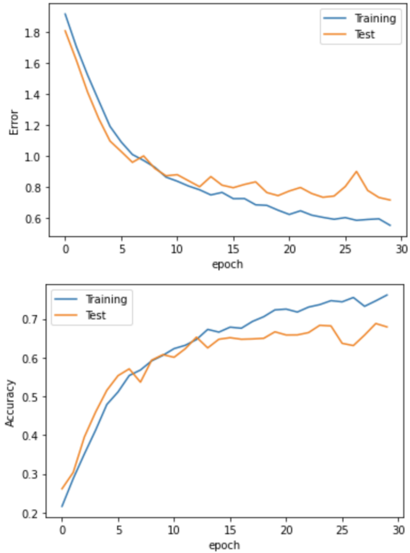
```
adam = Adam(learning_rate=0.005)
model.compile(loss='categorical_crossentropy',
              optimizer=adam, metrics=['accuracy'])
model.summary()

# Now train the model
estimator = model.fit(trnInp, trnTrg,
                      epochs=30,
                      batch_size=80,
                      verbose=0,
                      validation_data=(tstInp, tstTrg),
                      callbacks=[CustomCallback()])
```

# Results

Accuracy and Loss function

- Train error function: 0.6
- Test error function: 0.8
- Train accuracy: 0.73
- Test accuracy: 0.65



# Examples of wrong predictions

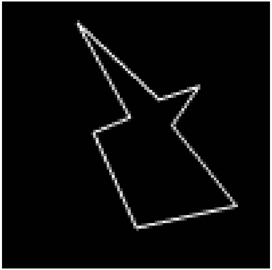


Figure: True: 8. Predicted:9

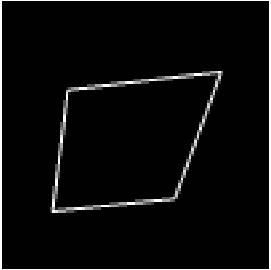


Figure: True: 4. Predicted:5

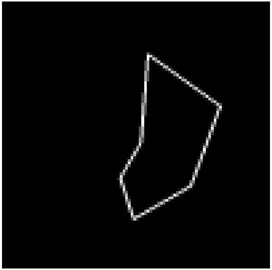


Figure: True: 6. Predicted:5

# Conclusion

Results:

- Best results obtained with only two convolutional blocks.
- Bigger size of the kernels in the CNN.
- Bigger size of the pooling (downsampling) to improve efficiency.
- Dropout on the input layer works better than in the hidden layers.

Possible improvements:

- Adding more filters and more hidden layers in the CNN
- Decreasing batch-size
- Increase the number of data (augmentation) by flipping or rotating images (any SO(n) matrix).