This document contains thought and decision making progress that I had while working on the Flatland project.
First thing to note is that the input data are simple geometric shapes so it is very simple to create more of it by manipulating the corresponding matrices (transposing, flipping symmetrically along the x and y axes) so that is the first thing that I did before trying any of the model architectures and quadrupled the amount of training samples.
As a starting point I used this CNN architecture:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=[50, 50, 1]))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(5, activation='softmax'))
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.Adam(),
              metrics=["accuracy"])
```

since this was the model with the best results that was shown during the lectures, the above model is called CNN1 in the repository. The model has 100% accuracy on the train data and, if I remember correctly, >95% on both test sets. The data augmentation trick worked really well, as without it the model has only 98% accuracy on the train set.
So the problem was solved at this point, however the model itself is quite big, it has 2,185,989 parameters, which is definitely overkill for a simple problem such as this.
Thus the next step was to make the model smaller. After experimenting with reducing the amount of parameters in the layers and adding extra Max Pooling I got this model architecture:

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(5, 5), activation='relu', input_shape=[50, 50, 1]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(8, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(5, activation='softmax'))
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.Adam(),
              metrics=["accuracy"])
```

This model is called CNN4 in the repository, it is quite a bit smaller then CNN1 with 19,181 parameters and it also works better with 99.99% accuracy on both test sets. The better accuracy is probably due to the model being easier to train due to smaller size.
I wanted to reduce the size even further so I started experimenting with remov-

ing some of the layers. I found that it was not possible to remove one of the convolutional layers as I guess the network needs atleast 2 of them to be able to detect vertices. Next I tried to think about what the model actually does when classifying the shapes. I came up with the following interpretation: the convolutional layers detect the vertices and then the dense layers count how many vertices were detected. With this interpretation it is quite obvious that the additional hidden dense layer is unnecessary and having only the output layer should work just as well. However when training a model without the additional dense layer I ran into the issue that the model would reach around 95% accuracy and then it would drop and never recover. After thinking about it I came up with the hypothesis that this is probably due to the fact that I was using the reLu activation function in my convolutional layers which is unbounded. So after training a while the convolutional layers would get so confident at classifying vertices and output such a high value that the dense layer would start double counting vertices and this would tank performance, thus the additional dense layer was being used to normalize the output of the convolutional layers. My hypothesis turned out to be correct as after changing the activation function to a sigmoid, which is bounded, I got a well functioning model that I was able to train successfully. This is the final architecture:

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(5, 5), activation='relu', input_shape=[50, 50, 1]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(8, kernel_size=(3, 3), activation='sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(5, activation='softmax'))
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.Adam(),
              metrics=["accuracy"])
model.summary()
```

It is called CNN5 in the repository, it is half as big as CNN4 with 10,285 parameters and works just as well. In conclusion I believe that CNN5 is pretty much optimal in terms of size and performance for this task as removing one of the convolutional layers would reduce performance.