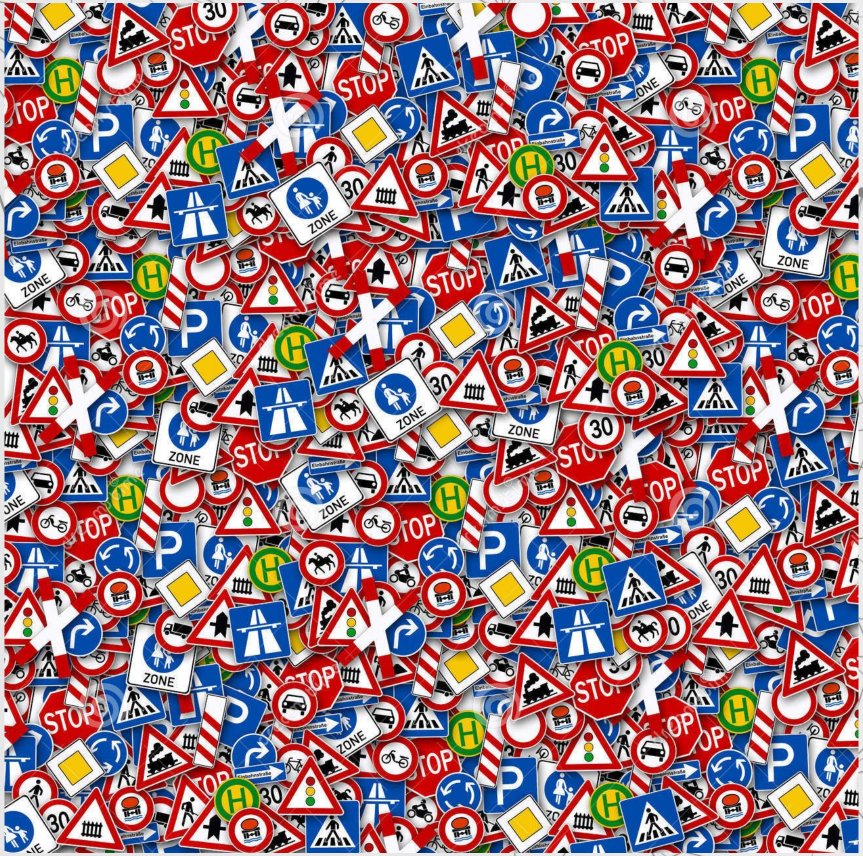
# FFNN for computer vision

Miguel Pérez
David Burrell
Ignacio Martínez

# 1st Case: Traffic Sign recognition



# Traffic Sign Recognition

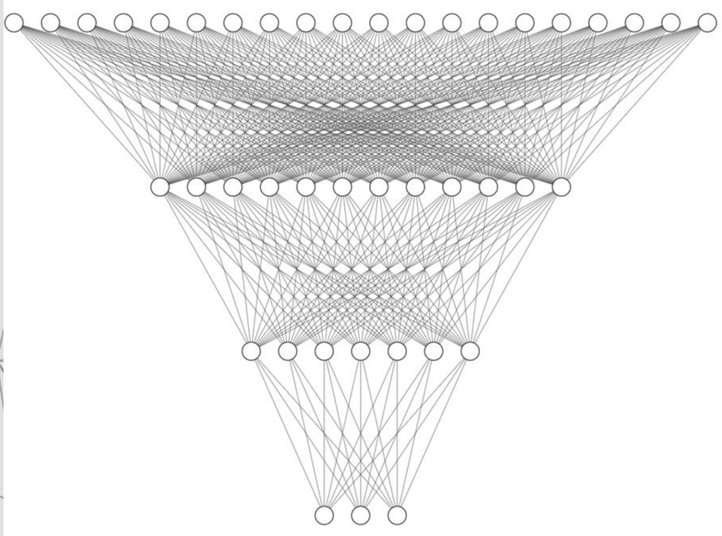
This task involved developing a neural net that can classify images of traffic signs

Image preprocessing performed:

1. Images are **loaded** from a .csv then **rescaled**, **resized** and **stored** in a vector.
2. **Labels** are attached to their corresponding images
3. The data was **split** into two parts: **train** and **test**.
4. The **train** was divided again into two parts: **train** and **validation.**
5. The tensor values are **normalized** from the range 0 - 255 to 0 - 1
6. The **numerical values** representing each traffic sign are transformed into a binary categorical value.

Once this is done, the data is ready to be fed to the Neural Network

# Architecture



Input layer: 150528

1st Hidden Layer: 1000

2nd Hidden Layer: 750

Output Layer: 43

```
data = mlp.fit(tr_signs,
               tr_labels,
               batch_size=25,
               epochs=150,
               verbose=1,
               validation_data=(va_signs, va_labels),
               callbacks=[tensorboard])
```

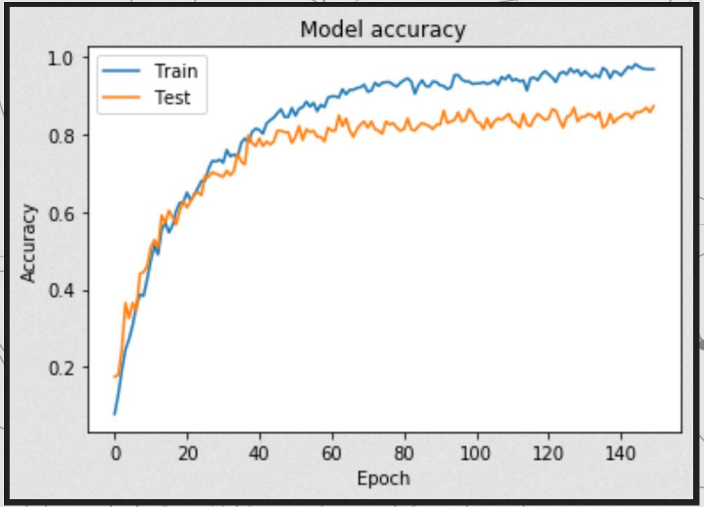
```
opt = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.99, nesterov=True)

mlp.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
mlp.summary()
```

```
==================================
Total params: 151,312,043
Trainable params: 151,312,043
Non-trainable params: 0
```

# Results

```
ffNN took 0.19642043113708496 seconds
Test loss: 0.4035492646727205 - Accuracy: 0.9085872577828383
```




# 2nd Case:
# General Purpose Recognition

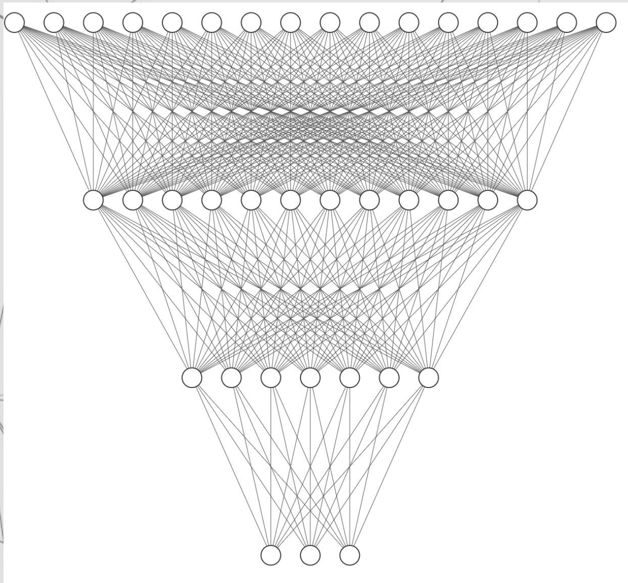
# Designed NN

We developed a FFNN trying to achieve the best score (accuracy)

We performed some steps on several trials and selected the values that gave us better accuracy.

1. We modified **number of layers** and we changed the **size of each layer**.
2. We varied the **activation function** for the inner layers. The best one was the relu.
3. We tried adding **weight initializers.** However, none of them improved the accuracy.
4. We tested **batch normalization** to the output of each layer to normalize the output of each layer. Again, this measure did not improved the performance
5. We used **Dropout** regularization to reduce overfitting. We applied values of 0.5 and 0.2.
6. We used the following optimizer:
7. optimizers.**SGD**(lr=0.001, decay=1e-6, momentum=0.9. **nesterov=True**)
8. We used as loss function the **categorical crosentropy** and **accuracy** as the metric.

# Architecture



Input layer: 3072

1st Hidden Layer: 1000

2nd Hidden Layer: 750

Output Layer: 43

```
data = mlp.fit(x_train, tr_labels,
               batch_size=32,
               epochs=100,
               verbose=1,
               validation_data=(x_val, va_labels),
               callbacks=[tensorboard])
```

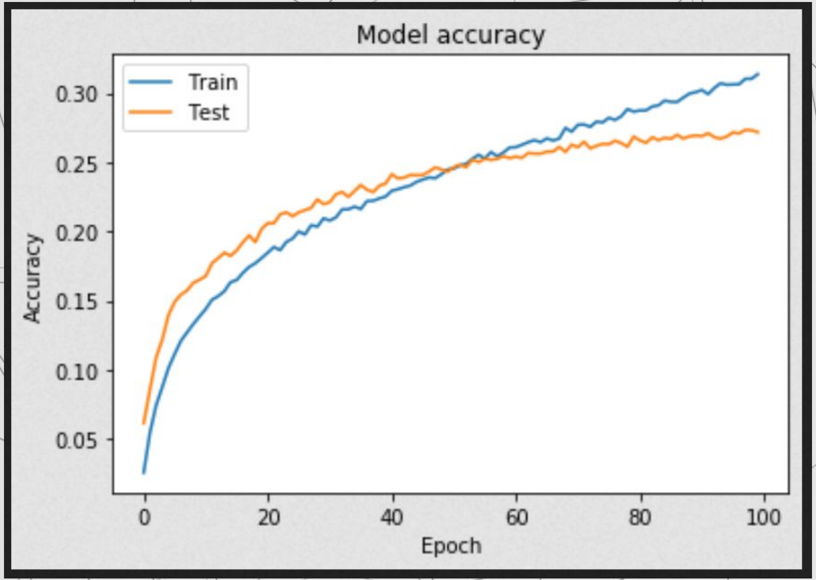
```
opt = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.99, nesterov=True)

mlp.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
mlp.summary()
```

```
================================
Total params: 3,898,850
Trainable params: 3,898,850
Non-trainable params: 0
________________________________
```

# Results

```
ffNN took 0.5458638668060303 seconds
Test loss: 3.007897315597534 - Accuracy: 0.2825
```




Took around 6 minutes and 30 seconds to train!

# Thanks for your attention!