W4995 Applied Machine Learning

Keras & Convolutional Neural Nets

04/17/19

Andreas C. Müller

Introduction to Keras

Keras Sequential

```
from keras.models import Sequential
from keras.layers import Dense, Activation
```

```
Using TensorFlow backend.
```

```
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax')])

# or
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))

# or
model = Sequential([
    Dense(32, input_shape=(784,), activation='relu'),
    Dense(10, activation='softmax')])
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_165 (Dense)            (None, 32)                25120
_________________________________________________________________
activation_113 (Activation)  (None, 32)                0
_________________________________________________________________
dense_166 (Dense)            (None, 10)                330
_________________________________________________________________
activation_114 (Activation)  (None, 10)                0
=================================================================
Total params: 25,450.0
Trainable params: 25,450.0
Non-trainable params: 0.0
_________________________________________________________________
```

Setting Optimizer

compile

```
compile(self, optimizer, loss, metrics=None, sample_weight_mode=None)
```

Configures the learning process.

Arguments

- **optimizer**: str (name of optimizer) or optimizer object. See optimizers.
- **loss**: str (name of objective function) or objective function. See objectives.
- **metrics**: list of metrics to be evaluated by the model during training and testing. Typically you will use `metrics=['accuracy']`. See metrics.

```
model.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
```

Training the model

```
fit(self, x, y, batch_size=32, epochs=10, verbose=1, callbacks=None, validation_split=0.0, validatio
```

6 / 46

Preparing MNIST data

```
from keras.datasets import mnist
import keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

num_classes = 10
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
60000 train samples

10000 test samples
```

Fit Model

```
model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1)
```

```
Epoch 1/10
60000/60000 [==============================] - 3s - loss: 0.4897 - acc: 0.8680
Epoch 2/10
```

Fit with Validation

```
model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1,
          validation_split=.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 2s - loss: 0.5146 - acc: 0.8616 - val_loss: 0.2425 - val_acc: 0.9322
```

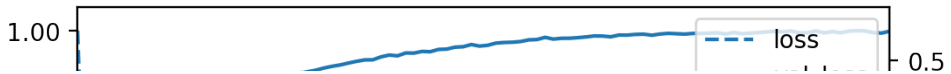
Evaluating on Test Set

```
score = model.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.120
Test Accuracy: 0.966
```

Loggers and Callbacks

```
history_callback = model.fit(X_train, y_train, batch_size=128,
                             epochs=100, verbose=1, validation_split=.1)
pd.DataFrame(history_callback.history).plot()
```



Wrappers for sklearn

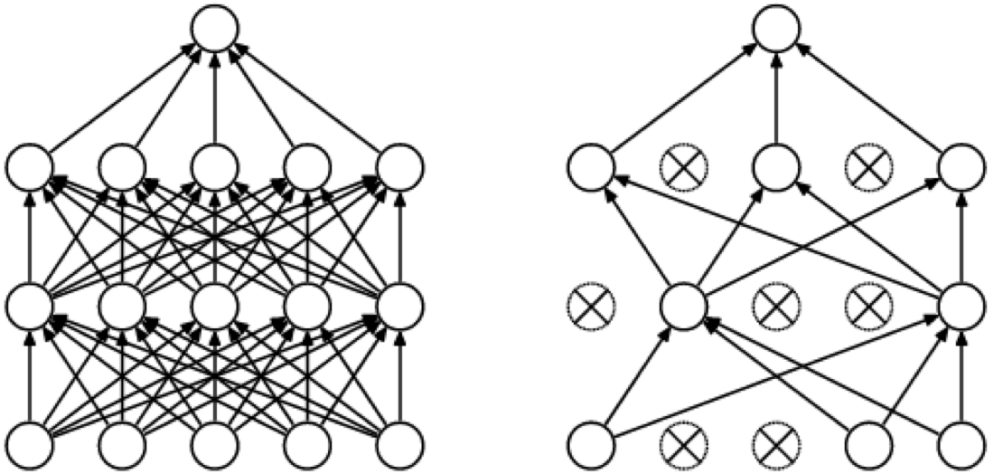
```
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from sklearn.model_selection import GridSearchCV

def make_model(optimizer="adam", hidden_size=32):
    model = Sequential([
        Dense(hidden_size, input_shape=(784,)),
        Activation('relu'),
        Dense(10),
        Activation('softmax'),
    ])
    model.compile(optimizer=optimizer,loss="categorical_crossentropy",
                  metrics=['accuracy'])
    return model

clf = KerasClassifier(make_model)
param_grid = {'epochs': [1, 5, 10],  # epochs is fit parameter, not in make_model!
              'hidden_size': [32, 64, 256]}
grid = GridSearchCV(clf, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

```
res = pd.DataFrame(grid.cv_results_)
res.pivot_table(index=["param_epochs", "param_hidden_size"],
                values=['mean_train_score', "mean_test_score"])
```

		mean_test_score	mean_train_score

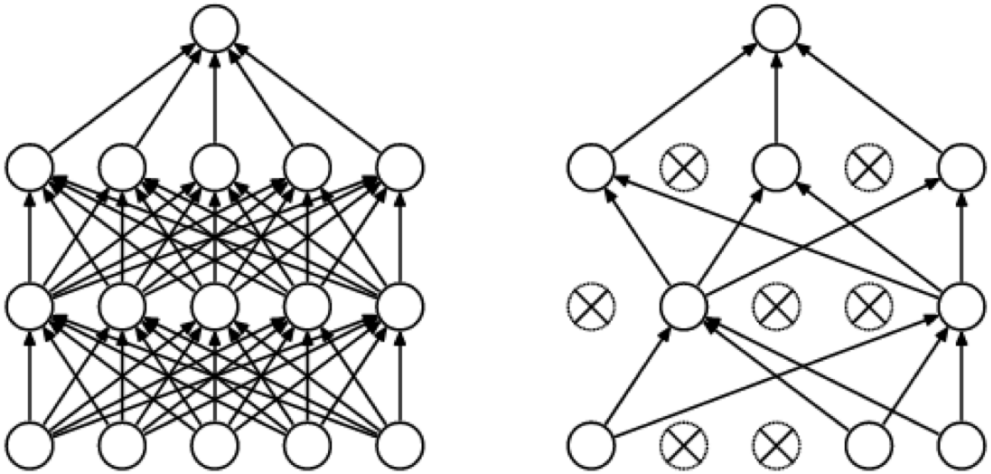
Drop-out

Drop-out Regularization



(a) Standard Neural Net (b) After applying dropout.

Drop-out Regularization



(a) Standard Neural Net (b) After applying dropout.

- https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
- Rate often as high as .5, i.e. 50% of units set to zero!
- Predictions: use all weights, down-weight by 1 - dropout rate

Ensemble Interpretation

- Every possible configuration represents different network.
- With p=.5 we jointly learn $\binom{n}{\frac{n}{2}}$ networks
- Networks share weights
- For last layer dropout: prediction is approximate geometric mean of predictions of sub-networks.

Implementing Drop-Out

```
from keras.layers import Dropout

model_dropout = Sequential([
    Dense(1024, input_shape=(784,), activation='relu'),
    Dropout(.5),
    Dense(1024, activation='relu'),
    Dropout(.5),
    Dense(10, activation='softmax'),
])
model_dropout.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
history_dropout = model_dropout.fit(X_train, y_train, batch_size=128,
                                    epochs=20, verbose=1, validation_split=.1)
```

When to use drop-out

- Avoids overfitting
- Allows using much deeper and larger models
- Slows down training somewhat
- Wasn't able to produce better results on MNIST (I don't have a GPU) but should be possible

Convolutional neural networks

Idea

- Translation invariance
- Weight sharing

Definition of Convolution

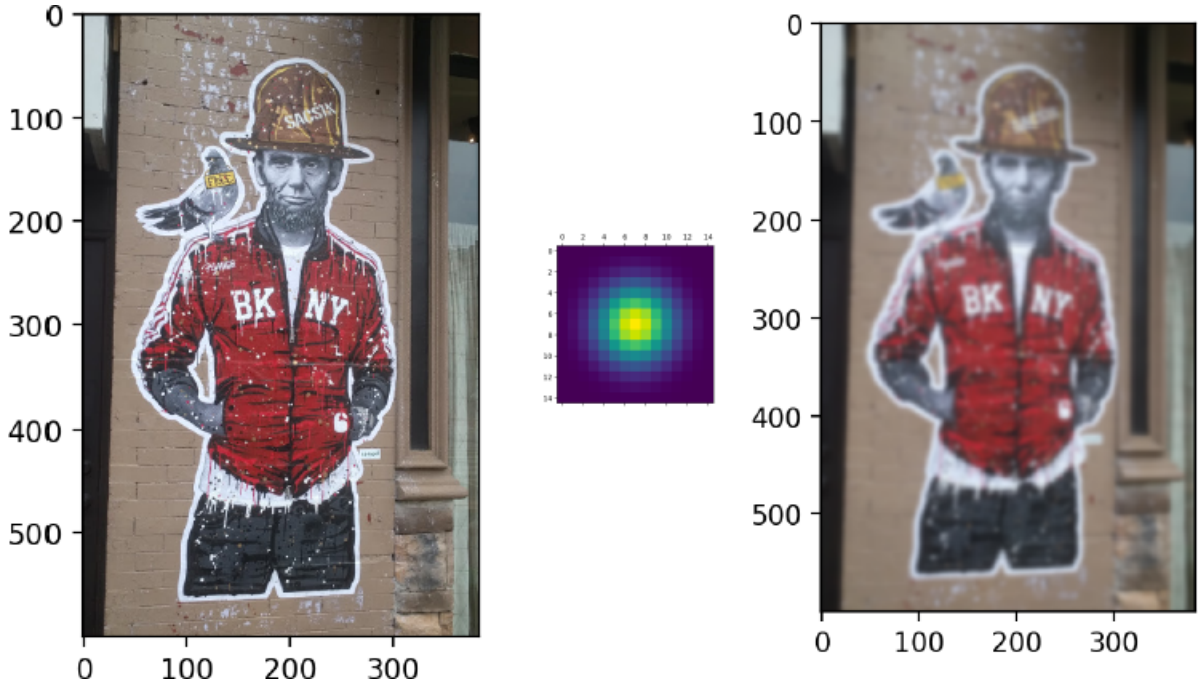
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

$$= \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

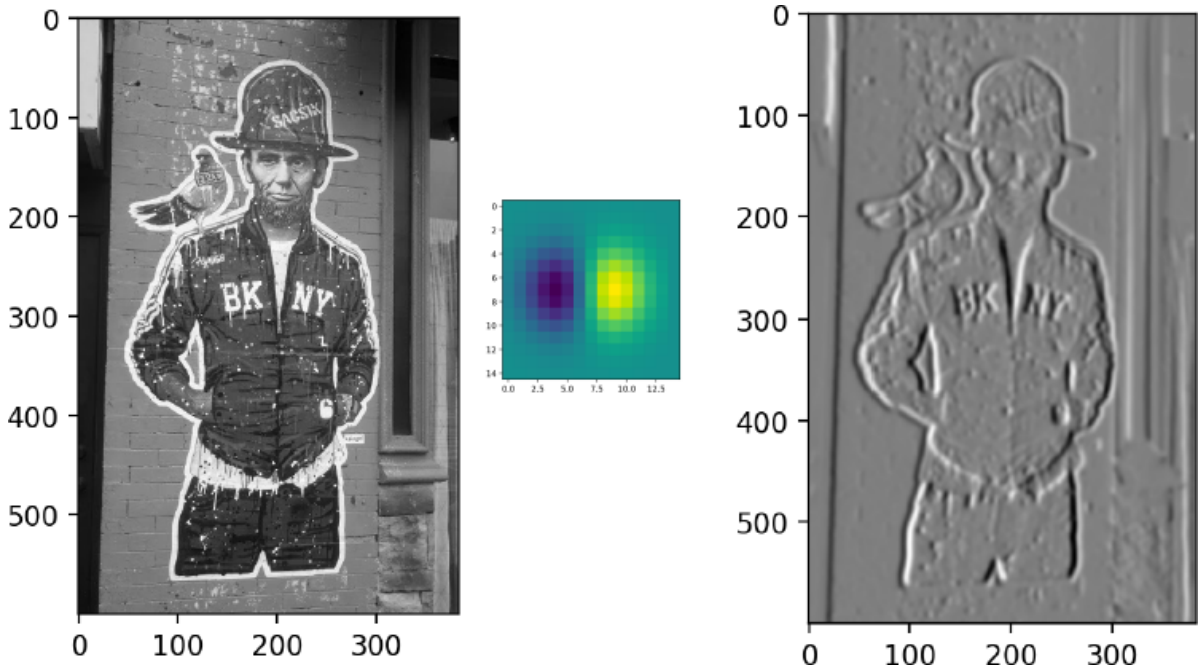
f 0 1 2 0 1 1 0 0 1 0

1d example: Gaussian smoothing

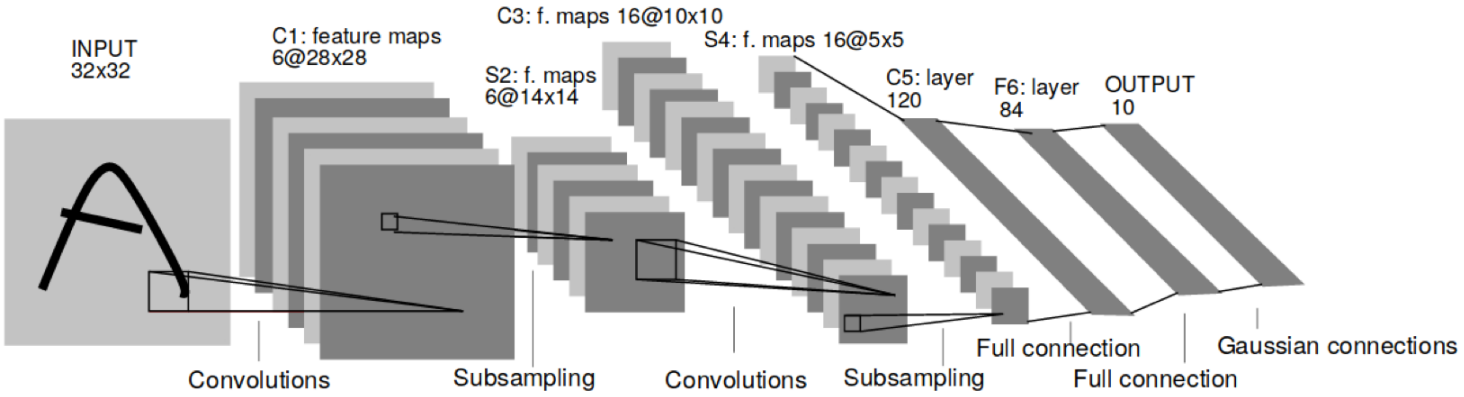
2d Smoothing



2d Gradients

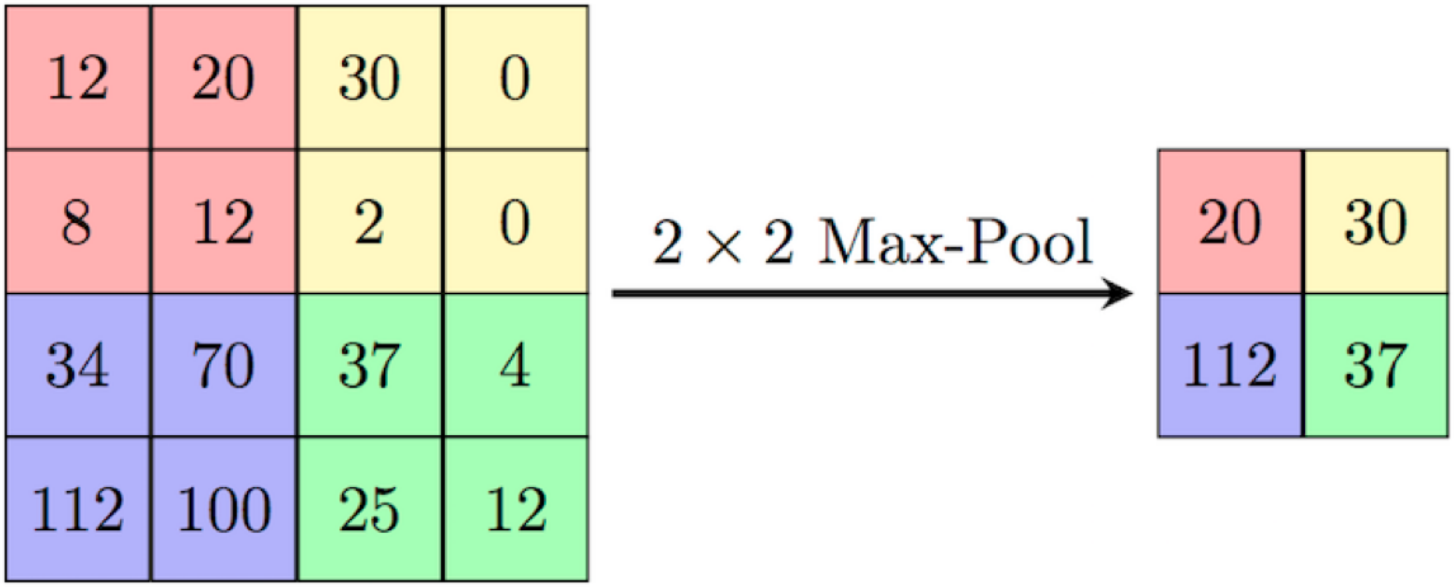


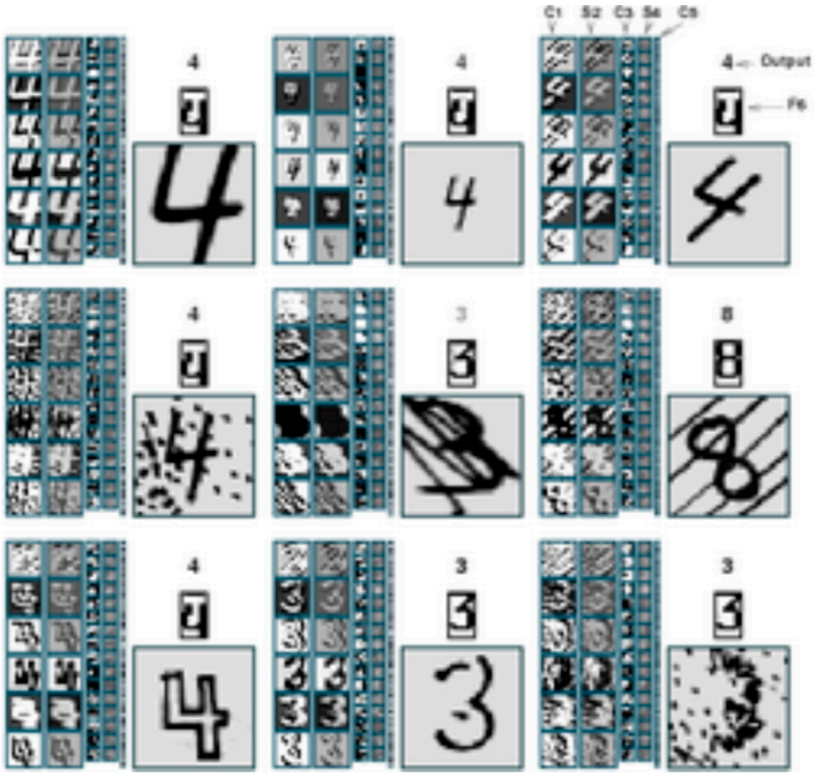
Convolution Neural Networks



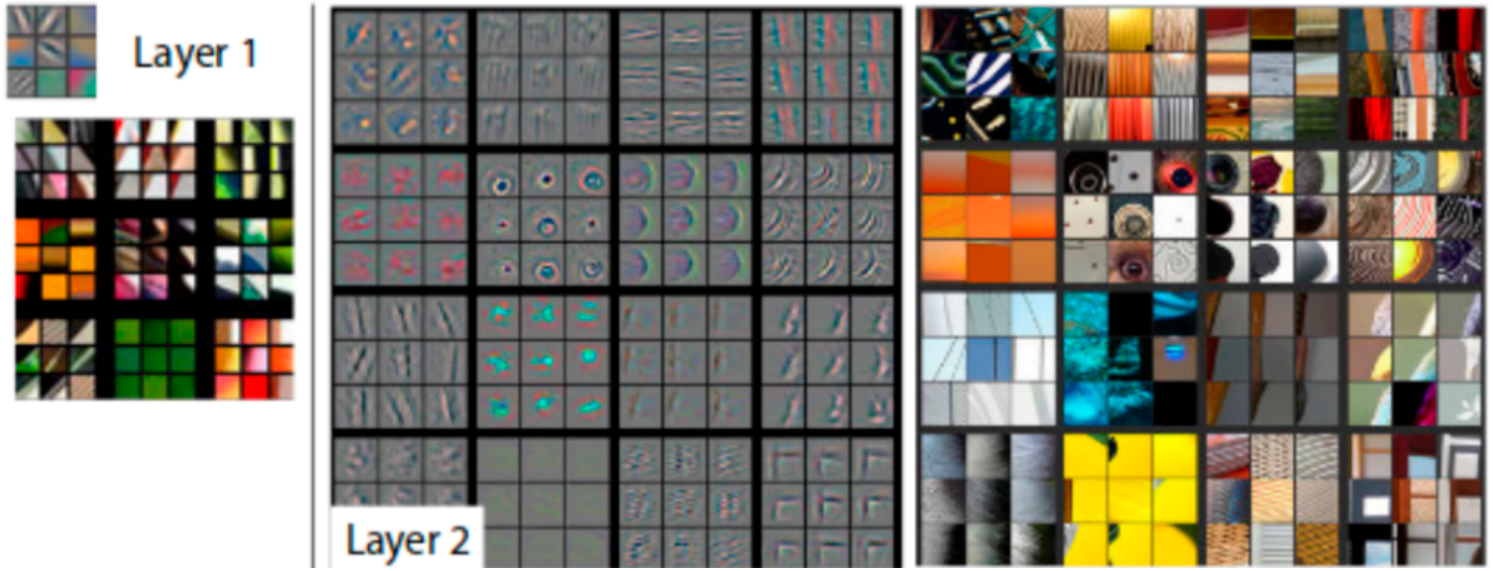
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner: Gradient-based learning applied to document recognition

Max Pooling

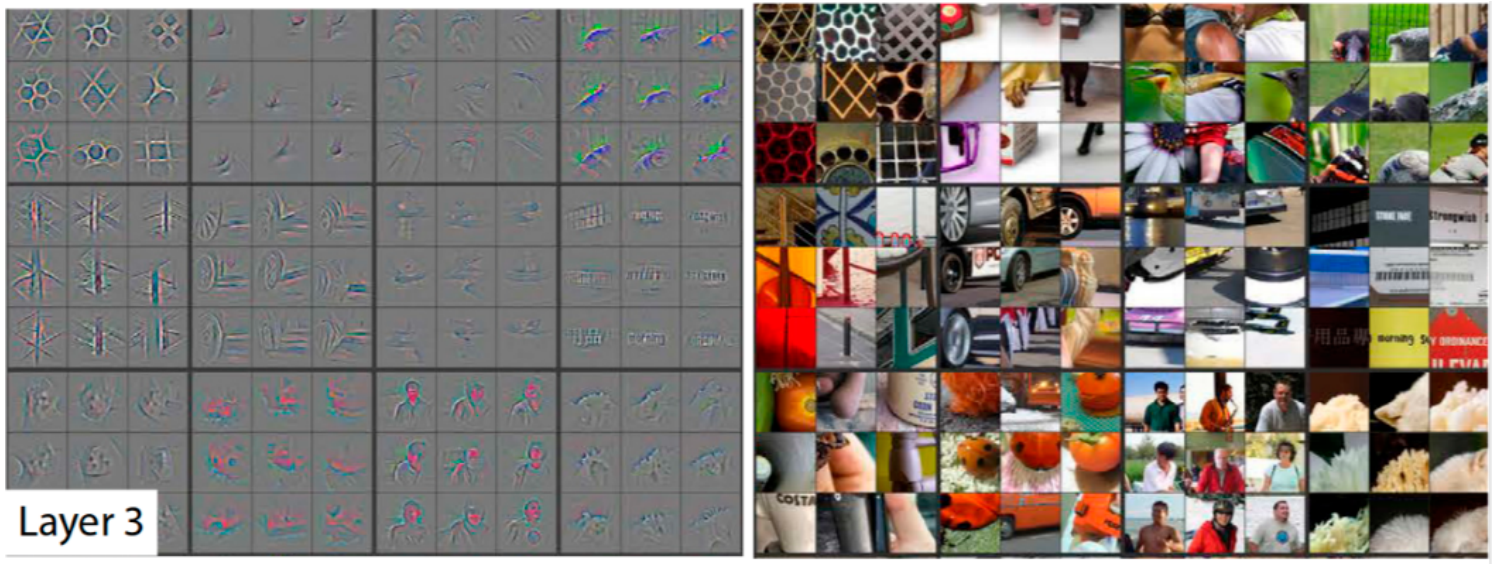


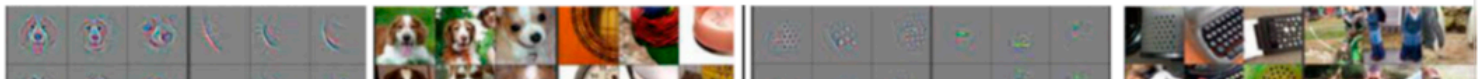

C1
S2
C3
S4
C5
4
Output
F6

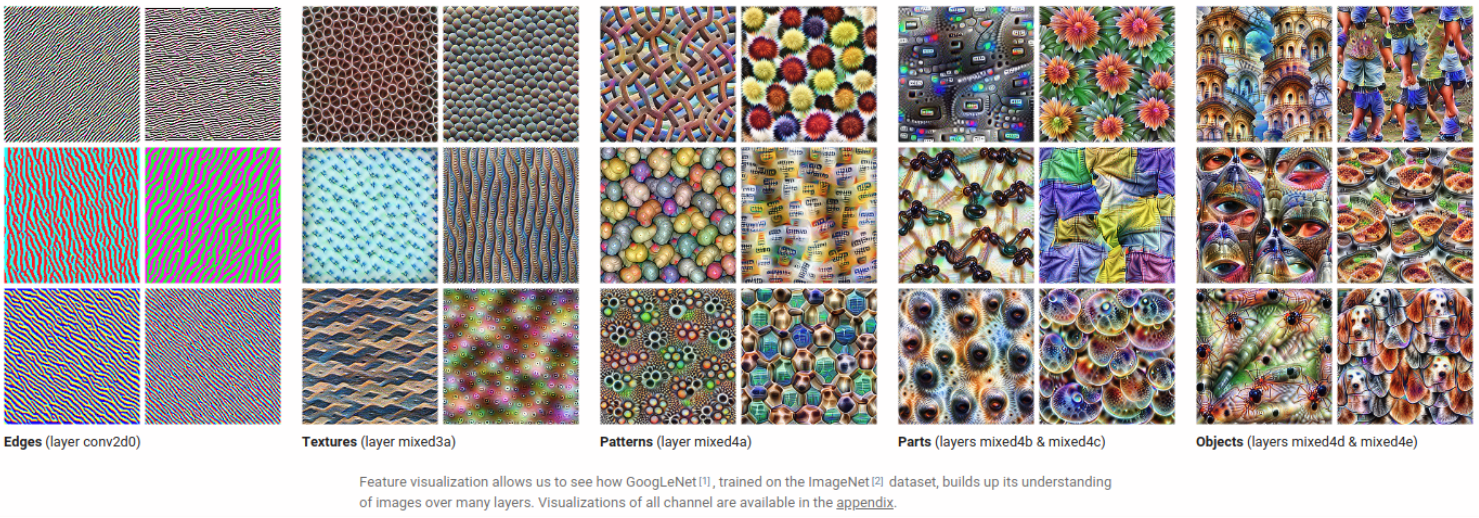
Deconvolution



https://www.cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf


Layer 3





https://distill.pub/2017/feature-visualization/

Alex Net

33 / 46

Conv-nets with keras

Preparing Data

```
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()


X_train_images = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
X_test_images = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Create Tiny Network

```
from keras.layers import Conv2D, MaxPooling2D, Flatten

num_classes = 10
cnn = Sequential()
cnn.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Conv2D(32, (3, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Flatten())
cnn.add(Dense(64, activation='relu'))
cnn.add(Dense(num_classes, activation='softmax'))
```

Number of Parameters

Convolutional Network for MNIST

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 11, 11, 32)        9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_163 (Dense)            (None, 64)                51264
_________________________________________________________________
dense_164 (Dense)            (None, 10)                650
=================================================================
Total params: 61,482.0
Trainable params: 61,482.0
Non-trainable params: 0.0
_________________________________________________________________
```

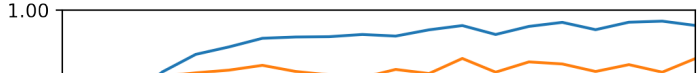
Dense Network for MNIST

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1024)              803840
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                10250
=================================================================
Total params: 1,863,690
Trainable params: 1,863,690
Non-trainable params: 0
_________________________________________________________________
```

Train and Evaluate

```
cnn.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
history_cnn = cnn.fit(X_train_images, y_train,
                      batch_size=128, epochs=20, verbose=1, validation_split=.1)
cnn.evaluate(X_test_images, y_test)
```

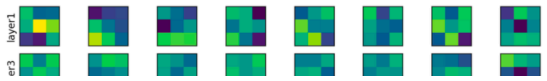
```
9952/10000 [============================>.] - ETA: 0s
[0.0890205832776292 53, 0.98429999999999995]
```

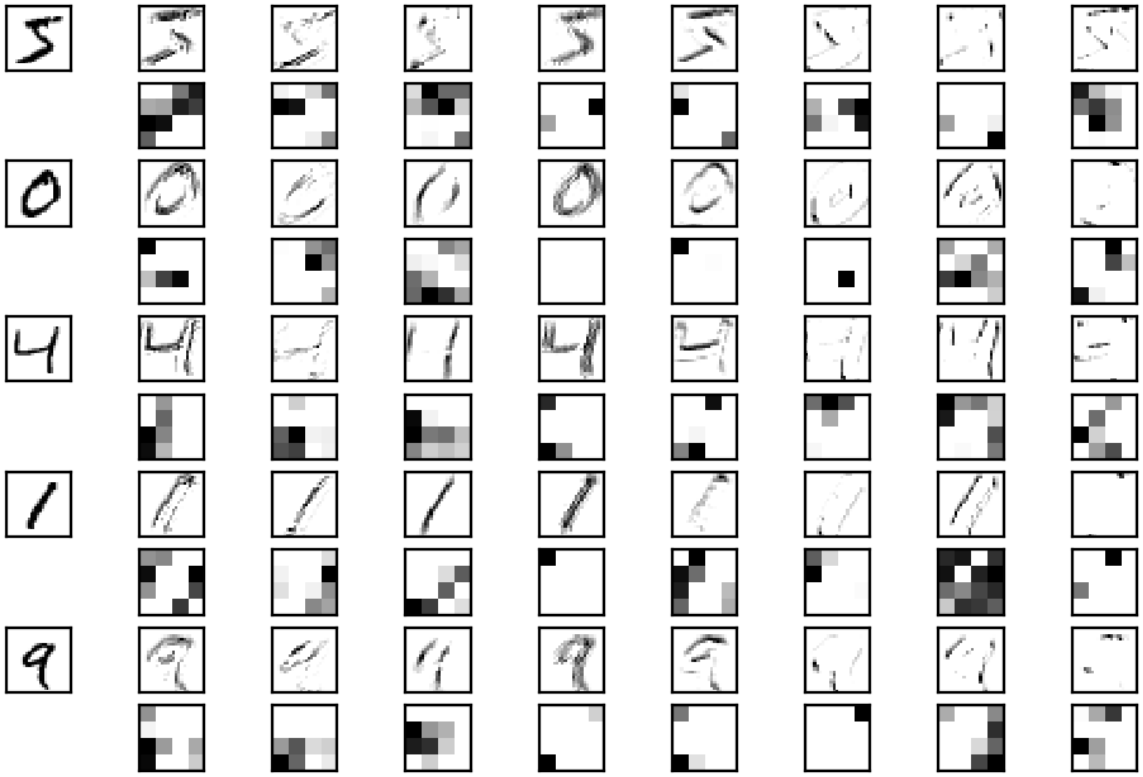


Visualize Filters

```
weights, biases = cnn_small.layers[0].get_weights()
weights2, biases2 = cnn_small.layers[2].get_weights()
print(weights.shape)
print(weights2.shape)
```

```
(3,3,1,8)
(3,3,8,8)
```





Batch Normalization

Batch Normalization

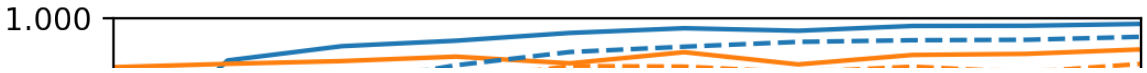
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;

Convnet with Batch Normalization

```
from keras.layers import BatchNormalization

num_class = 10
cnn_small_bn = Sequential()
cnn_small_bn.add(Conv2D(8, kernel_size=(3, 3),
                 input_shape=input_shape))
cnn_small_bn.add(Activation("relu"))
cnn_small_bn.add(BatchNormalization())
cnn_small_bn.add(MaxPooling2D(pool_size=(2, 2)))
cnn_small_bn.add(Conv2D(8, (3, 3)))
cnn_small_bn.add(Activation("relu"))
cnn_small_bn.add(BatchNormalization())
cnn_small_bn.add(MaxPooling2D(pool_size=(2, 2)))
cnn_small_bn.add(Flatten())
cnn_small_bn.add(Dense(64, activation='relu'))
cnn_small_bn.add(Dense(num_classes, activation='softmax'))
```

Learning speed and accuracy

For larger net (64 filters)



Questions ?