Demo: Convolutional neural networks on the “slash” dataset

Fraida Fund

In this demo, we’ll look at an example of a task that is difficult for “classical” machine learning models, and difficult for fully connected neural networks, but easy for convolutional neural networks.

```
import seaborn as sns
import matplotlib.pyplot as plt
import keras
import numpy as np
import pandas as pd
import scipy

from sklearn.model_selection import train_test_split
from sklearn import ensemble, neighbors, linear_model, svm

from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation, Conv2D, Flatten, BatchNormalization,
    InputLayer, AvgPool2D, MaxPool2D, GlobalAvgPool2D
import tensorflow.keras.backend as K
from keras.utils.vis_utils import plot_model
```

The slash dataset

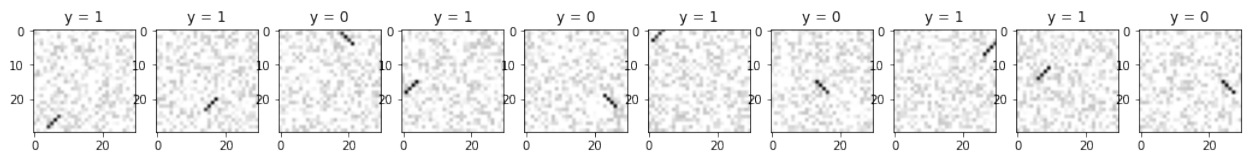
The “slash” dataset, developed by Sophie Searcy, is a set of images, each of which includes a “slash” on a background of random noise. The data is divided into two classes according to whether the slash is downward facing or upward facing.

```
def gen_example(size=20, label=0):

    max_s_pattern = int(size // 4)
    s_pattern = 4
    pattern = 1- np.eye(s_pattern)
    if label:
        pattern = pattern[:, ::-1]
    ex = np.ones((size,size))
    point_loc = np.random.randint(0, size - s_pattern + 1,
                                  size=(2, ))  # random x,y point
    ex[point_loc[0]:point_loc[0] + s_pattern, point_loc[1]:point_loc[1] +
       s_pattern] = pattern  # set point to
    ex = ex + .5*(np.random.rand(size, size) - .5)
    np.clip(ex,0.,1., out=ex)
    return ex
```

```
examples = []

n_side = 30
n_ex = 500 #number of examples in each class

for i in range(n_ex):
    examples.append(gen_example(size=n_side, label=0))
    examples.append(gen_example(size=n_side, label=1))
```

```
y = np.array([0,1]*n_ex)
x = np.stack(examples)
```

```
plt.figure(figsize=(18,4))

n_print = 10 # number of examples to show

ex_indices = np.random.choice(len(y), n_print, replace=False)
for i, index in enumerate(ex_indices):
    plt.subplot(1, n_print, i+1, )
    plt.imshow(x[index,...], cmap='gray')
    plt.title(f"y = {y[index]}")
```



We'l prepare training and test data in two formats:

- "flat" for traditional ML models and fully connected neural networks, which don't care about the spatial arrangement of the features.
- "image" for convolutional neural networks.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, test_size=0.25)

x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)

x_train_img = x_train[...,np.newaxis]
x_test_img = x_test[...,np.newaxis]
```

```
print("Flat data shape:  ", x_train_flat.shape)
print("Image data shape: ", x_train_img.shape)
```

```
Flat data shape:   (750, 900)
Image data shape:  (750, 30, 30, 1)
```

The feature data is in the range 0 to 1:

```
x.min(), x.max()
```

```
(0.0, 1.0)
```

Train logistic regression, random forest, KNN, SVM models

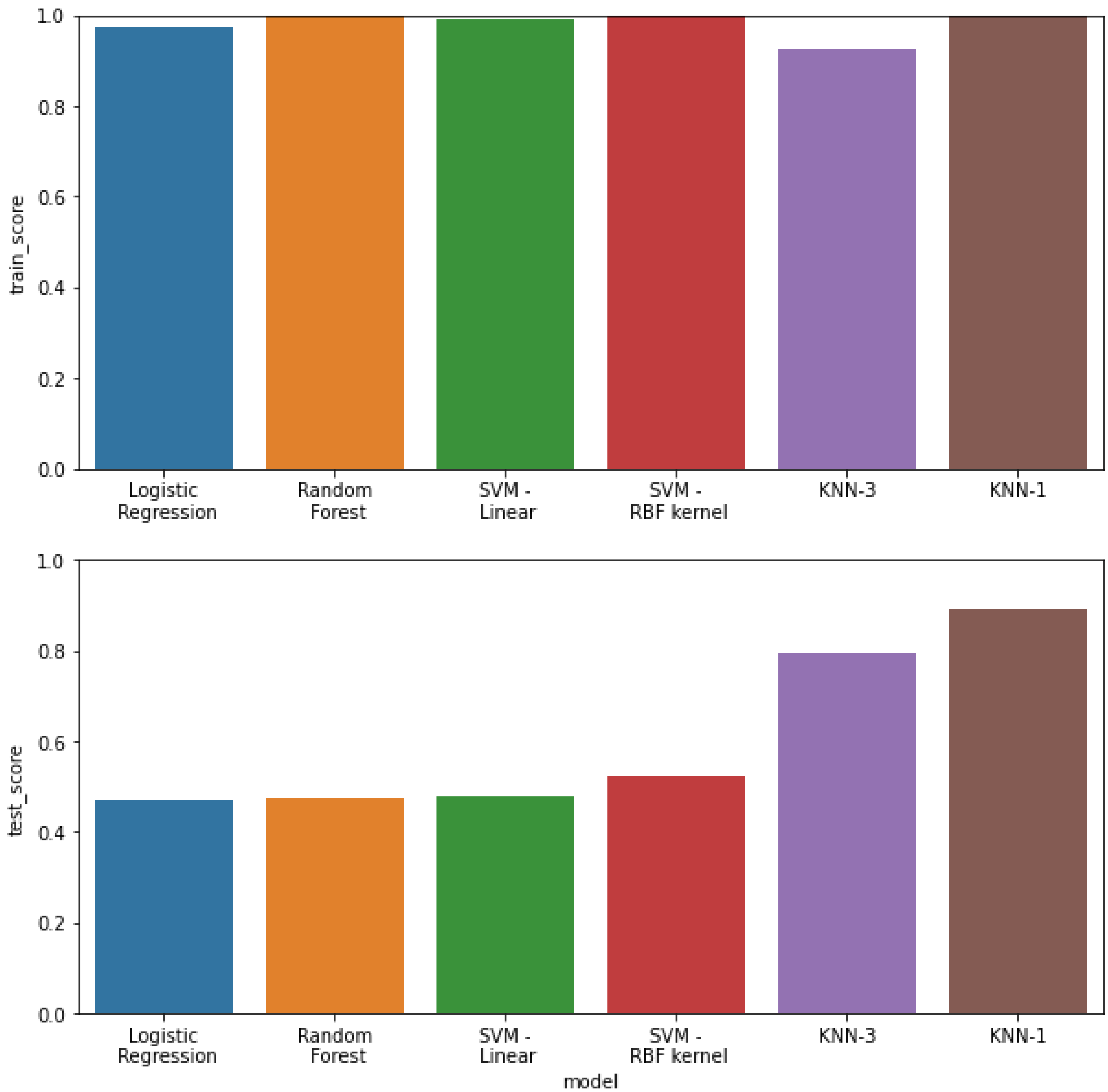
Next, we'll try to train some classic ML models on this dataset.

```
models = {
    "Logistic\n Regression": linear_model.LogisticRegression(),
    "KNN-1": neighbors.KNeighborsClassifier(n_neighbors=1),
    "KNN-3": neighbors.KNeighborsClassifier(n_neighbors=3),
```

```
    "Random\n Forest": ensemble.RandomForestClassifier(n_estimators=100),
    "SVM -\n Linear": svm.SVC(kernel="linear"),
    "SVM -\n RBF kernel": svm.SVC(kernel="rbf")
}
```

```
results = []

for model_name in models.keys():
    model = models[model_name]
    model.fit(x_train_flat, y_train)

    train_score = model.score(x_train_flat, y_train)
    test_score = model.score(x_test_flat, y_test)

    results.append({"model": model_name, "train_score": train_score, "test_score":
        test_score})
```

```
results_df = pd.DataFrame(results)

plt.figure(figsize =(10,10));

plt.subplot(2,1,1)
sns.barplot(x=results_df.sort_values('test_score')['model'],
    y=results_df.sort_values('test_score')['train_score']);
plt.ylim(0,1);
plt.xlabel("")

plt.subplot(2,1,2)
sns.barplot(x=results_df.sort_values('test_score')['model'],
    y=results_df.sort_values('test_score')['test_score']);
plt.ylim(0,1);
```



Are these the results we expected? Why or why not?

Do *any* of these models do a good job of learning whether a slash is forward-facing or backward-facing?

Train a fully connected neural network

```
nin = x_train_flat.shape[1]
nh1 = 64
nh2 = 64
nh3 = 64
nout = 1
model_fc = Sequential()
model_fc.add(Dense(units=nh1, input_shape=(nin,), activation='relu', name='hidden1'))
model_fc.add(Dense(units=nh2, input_shape=(nh1,), activation='relu', name='hidden2'))
model_fc.add(Dense(units=nh3, input_shape=(nh2,), activation='relu', name='hidden3'))
```

```python
model_fc.add(Dense(units=nout, activation='sigmoid', name='output'))

model_fc.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model_fc.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden1 (Dense)              (None, 64)                57664
_________________________________________________________________
hidden2 (Dense)              (None, 64)                4160
_________________________________________________________________
hidden3 (Dense)              (None, 64)                4160
_________________________________________________________________
output (Dense)               (None, 1)                 65
=================================================================
Total params: 66,049
Trainable params: 66,049
Non-trainable params: 0
_________________________________________________________________
```

```python
hist = model_fc.fit(x_train_flat, y_train, epochs=100,
     validation_split=0.25,  callbacks=[
        keras.callbacks.ReduceLROnPlateau(factor=.5, patience=2, verbose=1),
        keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True, verbose=1)
    ])
```

```
Epoch 1/100
18/18 [==============================] - 1s 16ms/step - loss: 0.7078 - accuracy: 0.4893 -
    val_loss: 0.6944 - val_accuracy: 0.5213
Epoch 2/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6989 - accuracy: 0.4858 -
    val_loss: 0.6959 - val_accuracy: 0.4734
Epoch 3/100
18/18 [==============================] - 0s 4ms/step - loss: 0.7029 - accuracy: 0.4893 -
    val_loss: 0.6948 - val_accuracy: 0.4255

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 4/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6955 - accuracy: 0.4929 -
    val_loss: 0.6965 - val_accuracy: 0.4787
Epoch 5/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6935 - accuracy: 0.4947 -
    val_loss: 0.6935 - val_accuracy: 0.5213
Epoch 6/100
18/18 [==============================] - 0s 3ms/step - loss: 0.7022 - accuracy: 0.4644 -
    val_loss: 0.6947 - val_accuracy: 0.5213
Epoch 7/100
18/18 [==============================] - 0s 5ms/step - loss: 0.7075 - accuracy: 0.4822 -
    val_loss: 0.6924 - val_accuracy: 0.5213
```

```
Epoch 8/100
18/18 [==============================] - 0s 5ms/step - loss: 0.7012 - accuracy: 0.4911 -
    val_loss: 0.6917 - val_accuracy: 0.5213
Epoch 9/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6988 - accuracy: 0.5142 -
    val_loss: 0.6985 - val_accuracy: 0.4787
Epoch 10/100
18/18 [==============================] - 0s 7ms/step - loss: 0.6942 - accuracy: 0.5018 -
    val_loss: 0.6923 - val_accuracy: 0.5213

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.000250000118743628.
Epoch 11/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6943 - accuracy: 0.5142 -
    val_loss: 0.6940 - val_accuracy: 0.4787
Epoch 12/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6938 - accuracy: 0.4715 -
    val_loss: 0.6944 - val_accuracy: 0.4787

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 13/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6930 - accuracy: 0.5071 -
    val_loss: 0.6946 - val_accuracy: 0.4787
Epoch 14/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6933 - accuracy: 0.5071 -
    val_loss: 0.6943 - val_accuracy: 0.4787

Epoch 00014: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
Epoch 15/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6930 - accuracy: 0.5036 -
    val_loss: 0.6938 - val_accuracy: 0.4628
Epoch 16/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6929 - accuracy: 0.5089 -
    val_loss: 0.6942 - val_accuracy: 0.4787

Epoch 00016: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
Epoch 17/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6929 - accuracy: 0.5071 -
    val_loss: 0.6940 - val_accuracy: 0.4787
Epoch 18/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6928 - accuracy: 0.5071 -
    val_loss: 0.6944 - val_accuracy: 0.4787

Epoch 00018: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
Epoch 19/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6928 - accuracy: 0.5071 -
    val_loss: 0.6944 - val_accuracy: 0.4787
Epoch 20/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6929 - accuracy: 0.5071 -
    val_loss: 0.6941 - val_accuracy: 0.4787

Epoch 00020: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
Epoch 21/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6928 - accuracy: 0.5071 -
    val_loss: 0.6942 - val_accuracy: 0.4787
```

```
Epoch 22/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6928 - accuracy: 0.5071 -
    val_loss: 0.6942 - val_accuracy: 0.4787

Epoch 00022: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.
Epoch 23/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6927 - accuracy: 0.5071 -
    val_loss: 0.6941 - val_accuracy: 0.4787
Epoch 24/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6927 - accuracy: 0.5071 -
    val_loss: 0.6942 - val_accuracy: 0.4787

Epoch 00024: ReduceLROnPlateau reducing learning rate to 1.9531250927684596e-06.
Epoch 25/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6927 - accuracy: 0.5071 -
    val_loss: 0.6941 - val_accuracy: 0.4787
Epoch 26/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6927 - accuracy: 0.5071 -
    val_loss: 0.6941 - val_accuracy: 0.4787

Epoch 00026: ReduceLROnPlateau reducing learning rate to 9.765625463842298e-07.
Epoch 27/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6927 - accuracy: 0.5071 -
    val_loss: 0.6941 - val_accuracy: 0.4787
Epoch 28/100
18/18 [==============================] - 0s 5ms/step - loss: 0.6927 - accuracy: 0.5071 -
    val_loss: 0.6941 - val_accuracy: 0.4787

Epoch 00028: ReduceLROnPlateau reducing learning rate to 4.882812731921149e-07.
Restoring model weights from the end of the best epoch.
Epoch 00028: early stopping
```

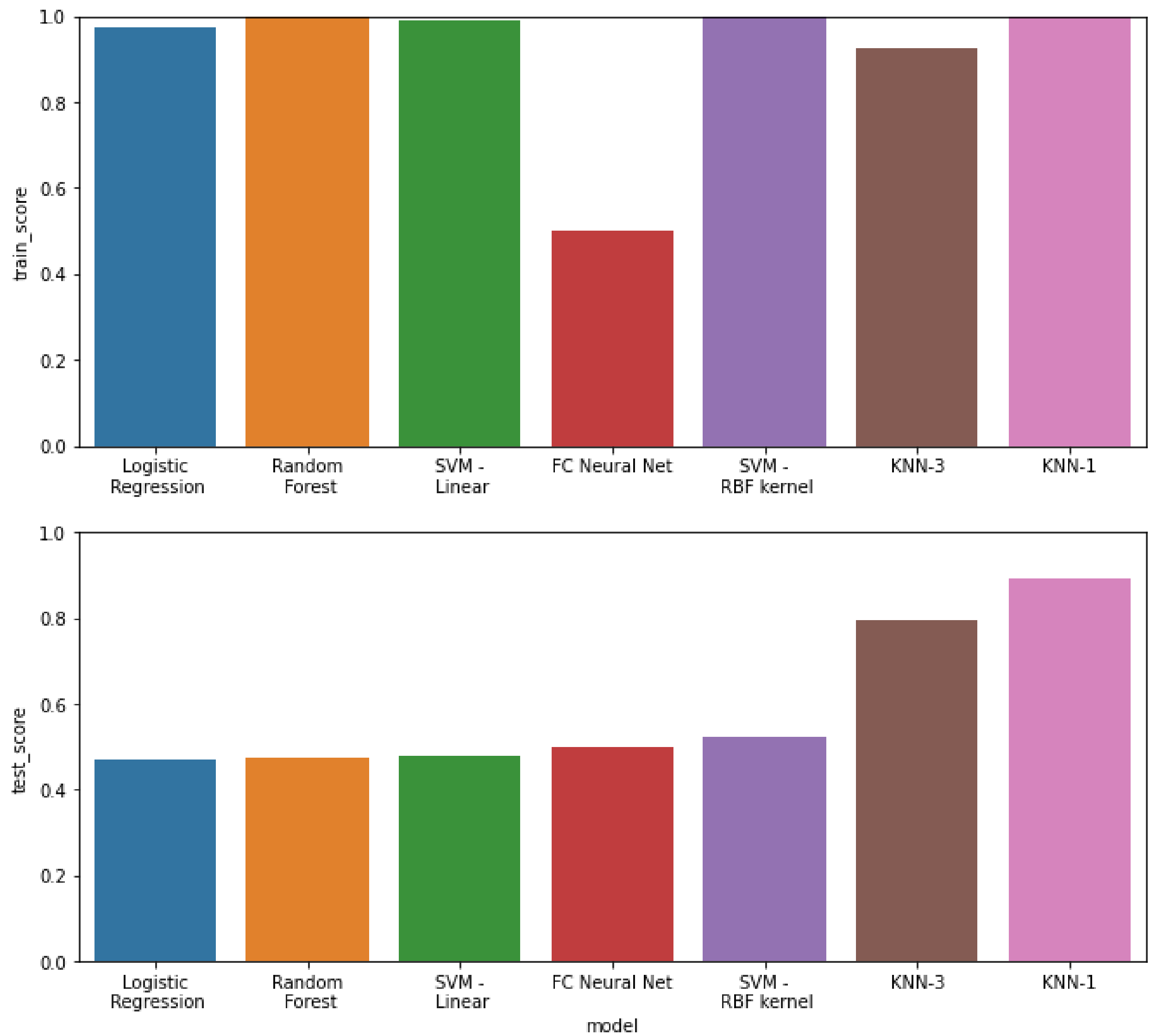
```
train_score = model_fc.evaluate(x_train_flat, y_train)[1]
test_score = model_fc.evaluate(x_test_flat, y_test)[1]
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.6938 - accuracy: 0.5000
8/8 [==============================] - 0s 2ms/step - loss: 0.6938 - accuracy: 0.5000
```

```
results.append({"model": 'FC Neural Net', "train_score": train_score, "test_score":
    test_score})
```

```
results_df = pd.DataFrame(results)

plt.figure(figsize =(11,10));

plt.subplot(2,1,1)
sns.barplot(x=results_df.sort_values('test_score')['model'],
    y=results_df.sort_values('test_score')['train_score']);
plt.ylim(0,1);
plt.xlabel("")

plt.subplot(2,1,2)
```

```
sns.barplot(x=results_df.sort_values('test_score')['model'],
    y=results_df.sort_values('test_score')['test_score']);
plt.ylim(0,1);
```



Train a convolutional neural network

```
filters = 10
model_conv = Sequential()
model_conv.add(InputLayer(input_shape=x_train_img.shape[1:]))
model_conv.add(Conv2D(filters, kernel_size=3, padding="same", activation="relu",
    use_bias=False ))
model_conv.add(MaxPool2D(pool_size=(2, 2)))
model_conv.add(BatchNormalization())
model_conv.add(Conv2D(filters, kernel_size=3, padding="same", activation="relu",
    use_bias=False ))
model_conv.add(GlobalAvgPool2D())
model_conv.add(Dense(1, activation="sigmoid"))
```

```python
model_conv.summary()

model_conv.compile("adam", loss="binary_crossentropy", metrics=["accuracy"])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 10)        90
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 10)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 15, 15, 10)        40
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 15, 15, 10)        900
_________________________________________________________________
global_average_pooling2d (Gl (None, 10)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 11
=================================================================
Total params: 1,041
Trainable params: 1,021
Non-trainable params: 20
_________________________________________________________________
```

```python
hist = model_conv.fit(x_train_img, y_train, epochs=100,
     validation_split=0.25,  callbacks=[
        keras.callbacks.ReduceLROnPlateau(factor=.5, patience=2, verbose=1),
        keras.callbacks.EarlyStopping(patience=20, restore_best_weights=True, verbose=1)
    ])

train_score = model_conv.evaluate(x_train_img, y_train)[1]
test_score = model_conv.evaluate(x_test_img, y_test)[1]
```

```
Epoch 1/100
18/18 [==============================] - 1s 28ms/step - loss: 0.7023 - accuracy: 0.5071 -
    val_loss: 0.6940 - val_accuracy: 0.4787
Epoch 2/100
18/18 [==============================] - 0s 15ms/step - loss: 0.6892 - accuracy: 0.5071 -
    val_loss: 0.6919 - val_accuracy: 0.5213
Epoch 3/100
18/18 [==============================] - 0s 13ms/step - loss: 0.6791 - accuracy: 0.5071 -
    val_loss: 0.6925 - val_accuracy: 0.5213
Epoch 4/100
18/18 [==============================] - 0s 12ms/step - loss: 0.6682 - accuracy: 0.5498 -
    val_loss: 0.6939 - val_accuracy: 0.5213

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 5/100
18/18 [==============================] - 0s 12ms/step - loss: 0.6574 - accuracy: 0.6263 -
    val_loss: 0.6940 - val_accuracy: 0.5213
Epoch 6/100
```

```
18/18 [==============================] - 0s 15ms/step - loss: 0.6464 - accuracy: 0.8060 -
    val_loss: 0.6948 - val_accuracy: 0.5213

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 7/100
18/18 [==============================] - 0s 13ms/step - loss: 0.6381 - accuracy: 0.9128 -
    val_loss: 0.6941 - val_accuracy: 0.5213
Epoch 8/100
18/18 [==============================] - 0s 12ms/step - loss: 0.6305 - accuracy: 0.9502 -
    val_loss: 0.6913 - val_accuracy: 0.5213
Epoch 9/100
18/18 [==============================] - 0s 13ms/step - loss: 0.6222 - accuracy: 0.9644 -
    val_loss: 0.6875 - val_accuracy: 0.5213
Epoch 10/100
18/18 [==============================] - 0s 11ms/step - loss: 0.6120 - accuracy: 0.9822 -
    val_loss: 0.6865 - val_accuracy: 0.5213
Epoch 11/100
18/18 [==============================] - 0s 11ms/step - loss: 0.6044 - accuracy: 0.9840 -
    val_loss: 0.6850 - val_accuracy: 0.5213
Epoch 12/100
18/18 [==============================] - 0s 14ms/step - loss: 0.5933 - accuracy: 0.9911 -
    val_loss: 0.6827 - val_accuracy: 0.5213
Epoch 13/100
18/18 [==============================] - 0s 11ms/step - loss: 0.5819 - accuracy: 0.9982 -
    val_loss: 0.6800 - val_accuracy: 0.5213
Epoch 14/100
18/18 [==============================] - 0s 14ms/step - loss: 0.5707 - accuracy: 0.9947 -
    val_loss: 0.6773 - val_accuracy: 0.5213
Epoch 15/100
18/18 [==============================] - 0s 11ms/step - loss: 0.5561 - accuracy: 1.0000 -
    val_loss: 0.6741 - val_accuracy: 0.5213
Epoch 16/100
18/18 [==============================] - 0s 12ms/step - loss: 0.5426 - accuracy: 1.0000 -
    val_loss: 0.6716 - val_accuracy: 0.5213
Epoch 17/100
18/18 [==============================] - 0s 13ms/step - loss: 0.5288 - accuracy: 1.0000 -
    val_loss: 0.6685 - val_accuracy: 0.5213
Epoch 18/100
18/18 [==============================] - 0s 12ms/step - loss: 0.5163 - accuracy: 1.0000 -
    val_loss: 0.6654 - val_accuracy: 0.5213
Epoch 19/100
18/18 [==============================] - 0s 12ms/step - loss: 0.4970 - accuracy: 1.0000 -
    val_loss: 0.6620 - val_accuracy: 0.5213
Epoch 20/100
18/18 [==============================] - 0s 15ms/step - loss: 0.4796 - accuracy: 1.0000 -
    val_loss: 0.6599 - val_accuracy: 0.5213
Epoch 21/100
18/18 [==============================] - 0s 14ms/step - loss: 0.4617 - accuracy: 1.0000 -
    val_loss: 0.6588 - val_accuracy: 0.5213
Epoch 22/100
18/18 [==============================] - 0s 10ms/step - loss: 0.4443 - accuracy: 1.0000 -
    val_loss: 0.6620 - val_accuracy: 0.5213
Epoch 23/100
18/18 [==============================] - 0s 10ms/step - loss: 0.4259 - accuracy: 1.0000 -
```

```
    val_loss: 0.6545 - val_accuracy: 0.5213
Epoch 24/100
18/18 [==============================] - 0s 17ms/step - loss: 0.4051 - accuracy: 1.0000 -
    val_loss: 0.6543 - val_accuracy: 0.5213
Epoch 25/100
18/18 [==============================] - 0s 15ms/step - loss: 0.3862 - accuracy: 1.0000 -
    val_loss: 0.6573 - val_accuracy: 0.5213
Epoch 26/100
18/18 [==============================] - 0s 11ms/step - loss: 0.3711 - accuracy: 1.0000 -
    val_loss: 0.6663 - val_accuracy: 0.5213

Epoch 00026: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 27/100
18/18 [==============================] - 0s 11ms/step - loss: 0.3556 - accuracy: 1.0000 -
    val_loss: 0.6576 - val_accuracy: 0.5213
Epoch 28/100
18/18 [==============================] - 0s 14ms/step - loss: 0.3457 - accuracy: 1.0000 -
    val_loss: 0.6448 - val_accuracy: 0.5213
Epoch 29/100
18/18 [==============================] - 0s 15ms/step - loss: 0.3368 - accuracy: 1.0000 -
    val_loss: 0.6263 - val_accuracy: 0.5213
Epoch 30/100
18/18 [==============================] - 0s 13ms/step - loss: 0.3266 - accuracy: 1.0000 -
    val_loss: 0.5917 - val_accuracy: 0.5213
Epoch 31/100
18/18 [==============================] - 0s 14ms/step - loss: 0.3197 - accuracy: 1.0000 -
    val_loss: 0.5843 - val_accuracy: 0.5213
Epoch 32/100
18/18 [==============================] - 0s 16ms/step - loss: 0.3103 - accuracy: 1.0000 -
    val_loss: 0.5723 - val_accuracy: 0.5213
Epoch 33/100
18/18 [==============================] - 0s 13ms/step - loss: 0.3023 - accuracy: 1.0000 -
    val_loss: 0.5468 - val_accuracy: 0.5213
Epoch 34/100
18/18 [==============================] - 0s 18ms/step - loss: 0.2956 - accuracy: 1.0000 -
    val_loss: 0.5415 - val_accuracy: 0.5213
Epoch 35/100
18/18 [==============================] - 0s 13ms/step - loss: 0.2886 - accuracy: 1.0000 -
    val_loss: 0.5182 - val_accuracy: 0.5213
Epoch 36/100
18/18 [==============================] - 0s 13ms/step - loss: 0.2811 - accuracy: 1.0000 -
    val_loss: 0.4976 - val_accuracy: 0.5266
Epoch 37/100
18/18 [==============================] - 0s 17ms/step - loss: 0.2757 - accuracy: 1.0000 -
    val_loss: 0.4772 - val_accuracy: 0.5479
Epoch 38/100
18/18 [==============================] - 0s 12ms/step - loss: 0.2652 - accuracy: 1.0000 -
    val_loss: 0.4455 - val_accuracy: 0.5904
Epoch 39/100
18/18 [==============================] - 0s 11ms/step - loss: 0.2564 - accuracy: 1.0000 -
    val_loss: 0.4329 - val_accuracy: 0.6117
Epoch 40/100
18/18 [==============================] - 0s 11ms/step - loss: 0.2544 - accuracy: 1.0000 -
    val_loss: 0.4131 - val_accuracy: 0.6596
```

```
Epoch 41/100
18/18 [==============================] - 0s 12ms/step - loss: 0.2468 - accuracy: 1.0000 -
    val_loss: 0.3756 - val_accuracy: 0.8138
Epoch 42/100
18/18 [==============================] - 0s 13ms/step - loss: 0.2392 - accuracy: 1.0000 -
    val_loss: 0.3384 - val_accuracy: 0.9468
Epoch 43/100
18/18 [==============================] - 0s 12ms/step - loss: 0.2334 - accuracy: 1.0000 -
    val_loss: 0.3255 - val_accuracy: 0.9574
Epoch 44/100
18/18 [==============================] - 0s 12ms/step - loss: 0.2303 - accuracy: 1.0000 -
    val_loss: 0.3038 - val_accuracy: 0.9787
Epoch 45/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2210 - accuracy: 1.0000 -
    val_loss: 0.2918 - val_accuracy: 0.9840
Epoch 46/100
18/18 [==============================] - 0s 12ms/step - loss: 0.2169 - accuracy: 1.0000 -
    val_loss: 0.2894 - val_accuracy: 0.9787
Epoch 47/100
18/18 [==============================] - 0s 12ms/step - loss: 0.2088 - accuracy: 1.0000 -
    val_loss: 0.2671 - val_accuracy: 0.9894
Epoch 48/100
18/18 [==============================] - 0s 14ms/step - loss: 0.2054 - accuracy: 1.0000 -
    val_loss: 0.2476 - val_accuracy: 0.9947
Epoch 49/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1976 - accuracy: 1.0000 -
    val_loss: 0.2423 - val_accuracy: 0.9947
Epoch 50/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1967 - accuracy: 1.0000 -
    val_loss: 0.2308 - val_accuracy: 1.0000
Epoch 51/100
18/18 [==============================] - 0s 15ms/step - loss: 0.1893 - accuracy: 1.0000 -
    val_loss: 0.2218 - val_accuracy: 1.0000
Epoch 52/100
18/18 [==============================] - 0s 18ms/step - loss: 0.1839 - accuracy: 1.0000 -
    val_loss: 0.2094 - val_accuracy: 1.0000
Epoch 53/100
18/18 [==============================] - 0s 15ms/step - loss: 0.1817 - accuracy: 1.0000 -
    val_loss: 0.2061 - val_accuracy: 1.0000
Epoch 54/100
18/18 [==============================] - 0s 16ms/step - loss: 0.1739 - accuracy: 1.0000 -
    val_loss: 0.1952 - val_accuracy: 1.0000
Epoch 55/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1657 - accuracy: 1.0000 -
    val_loss: 0.1926 - val_accuracy: 1.0000
Epoch 56/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1665 - accuracy: 1.0000 -
    val_loss: 0.1860 - val_accuracy: 1.0000
Epoch 57/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1604 - accuracy: 1.0000 -
    val_loss: 0.1744 - val_accuracy: 1.0000
Epoch 58/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1548 - accuracy: 1.0000 -
    val_loss: 0.1686 - val_accuracy: 1.0000
```

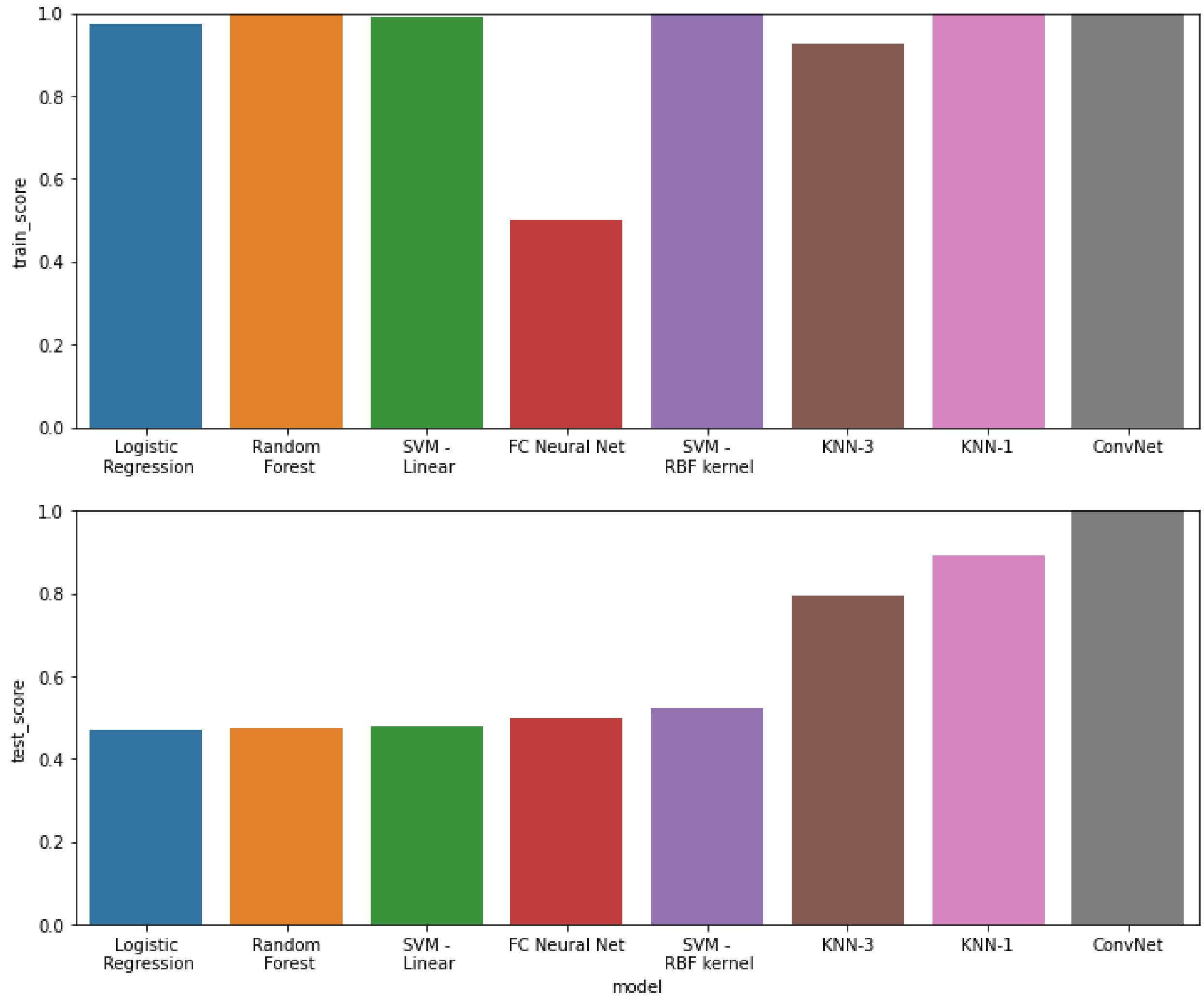
```
Epoch 59/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1543 - accuracy: 1.0000 -
    val_loss: 0.1643 - val_accuracy: 1.0000
Epoch 60/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1486 - accuracy: 1.0000 -
    val_loss: 0.1625 - val_accuracy: 1.0000
Epoch 61/100
18/18 [==============================] - 0s 11ms/step - loss: 0.1436 - accuracy: 1.0000 -
    val_loss: 0.1516 - val_accuracy: 1.0000
Epoch 62/100
18/18 [==============================] - 0s 11ms/step - loss: 0.1403 - accuracy: 1.0000 -
    val_loss: 0.1468 - val_accuracy: 1.0000
Epoch 63/100
18/18 [==============================] - 0s 16ms/step - loss: 0.1384 - accuracy: 1.0000 -
    val_loss: 0.1415 - val_accuracy: 1.0000
Epoch 64/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1369 - accuracy: 1.0000 -
    val_loss: 0.1404 - val_accuracy: 1.0000
Epoch 65/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1309 - accuracy: 1.0000 -
    val_loss: 0.1301 - val_accuracy: 1.0000
Epoch 66/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1283 - accuracy: 1.0000 -
    val_loss: 0.1266 - val_accuracy: 1.0000
Epoch 67/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1213 - accuracy: 1.0000 -
    val_loss: 0.1256 - val_accuracy: 1.0000
Epoch 68/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1216 - accuracy: 1.0000 -
    val_loss: 0.1249 - val_accuracy: 1.0000
Epoch 69/100
18/18 [==============================] - 0s 11ms/step - loss: 0.1201 - accuracy: 1.0000 -
    val_loss: 0.1157 - val_accuracy: 1.0000
Epoch 70/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1156 - accuracy: 1.0000 -
    val_loss: 0.1091 - val_accuracy: 1.0000
Epoch 71/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1116 - accuracy: 1.0000 -
    val_loss: 0.1084 - val_accuracy: 1.0000
Epoch 72/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1083 - accuracy: 1.0000 -
    val_loss: 0.1042 - val_accuracy: 1.0000
Epoch 73/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1085 - accuracy: 1.0000 -
    val_loss: 0.0996 - val_accuracy: 1.0000
Epoch 74/100
18/18 [==============================] - 0s 11ms/step - loss: 0.1012 - accuracy: 1.0000 -
    val_loss: 0.0934 - val_accuracy: 1.0000
Epoch 75/100
18/18 [==============================] - 0s 14ms/step - loss: 0.0951 - accuracy: 1.0000 -
    val_loss: 0.0862 - val_accuracy: 1.0000
Epoch 76/100
18/18 [==============================] - 0s 9ms/step - loss: 0.0921 - accuracy: 1.0000 -
    val_loss: 0.0821 - val_accuracy: 1.0000
```

```
Epoch 77/100
18/18 [==============================] - 0s 10ms/step - loss: 0.0925 - accuracy: 1.0000 -
    val_loss: 0.0787 - val_accuracy: 1.0000
Epoch 78/100
18/18 [==============================] - 0s 12ms/step - loss: 0.0859 - accuracy: 1.0000 -
    val_loss: 0.0772 - val_accuracy: 1.0000
Epoch 79/100
18/18 [==============================] - 0s 17ms/step - loss: 0.0813 - accuracy: 1.0000 -
    val_loss: 0.0781 - val_accuracy: 1.0000
Epoch 80/100
18/18 [==============================] - 0s 12ms/step - loss: 0.0773 - accuracy: 1.0000 -
    val_loss: 0.0805 - val_accuracy: 1.0000

Epoch 00080: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
Epoch 81/100
18/18 [==============================] - 0s 10ms/step - loss: 0.0761 - accuracy: 1.0000 -
    val_loss: 0.0782 - val_accuracy: 1.0000
Epoch 82/100
18/18 [==============================] - 0s 15ms/step - loss: 0.0737 - accuracy: 1.0000 -
    val_loss: 0.0766 - val_accuracy: 1.0000
Epoch 83/100
18/18 [==============================] - 0s 13ms/step - loss: 0.0753 - accuracy: 1.0000 -
    val_loss: 0.0750 - val_accuracy: 1.0000
Epoch 84/100
18/18 [==============================] - 0s 14ms/step - loss: 0.0713 - accuracy: 1.0000 -
    val_loss: 0.0738 - val_accuracy: 1.0000
Epoch 85/100
18/18 [==============================] - 0s 14ms/step - loss: 0.0718 - accuracy: 1.0000 -
    val_loss: 0.0736 - val_accuracy: 1.0000
Epoch 86/100
18/18 [==============================] - 0s 13ms/step - loss: 0.0708 - accuracy: 1.0000 -
    val_loss: 0.0739 - val_accuracy: 1.0000
Epoch 87/100
18/18 [==============================] - 0s 10ms/step - loss: 0.0679 - accuracy: 1.0000 -
    val_loss: 0.0729 - val_accuracy: 1.0000
Epoch 88/100
18/18 [==============================] - 0s 16ms/step - loss: 0.0692 - accuracy: 1.0000 -
    val_loss: 0.0707 - val_accuracy: 1.0000
Epoch 89/100
18/18 [==============================] - 0s 14ms/step - loss: 0.0651 - accuracy: 1.0000 -
    val_loss: 0.0697 - val_accuracy: 1.0000
Epoch 90/100
18/18 [==============================] - 0s 11ms/step - loss: 0.0678 - accuracy: 1.0000 -
    val_loss: 0.0692 - val_accuracy: 1.0000
Epoch 91/100
18/18 [==============================] - 0s 12ms/step - loss: 0.0646 - accuracy: 1.0000 -
    val_loss: 0.0661 - val_accuracy: 1.0000
Epoch 92/100
18/18 [==============================] - 0s 13ms/step - loss: 0.0614 - accuracy: 1.0000 -
    val_loss: 0.0657 - val_accuracy: 1.0000
Epoch 93/100
18/18 [==============================] - 0s 11ms/step - loss: 0.0612 - accuracy: 1.0000 -
    val_loss: 0.0653 - val_accuracy: 1.0000
Epoch 94/100
```

```
18/18 [==============================] - 0s 11ms/step - loss: 0.0613 - accuracy: 1.0000 -
    val_loss: 0.0638 - val_accuracy: 1.0000
Epoch 95/100
18/18 [==============================] - 0s 13ms/step - loss: 0.0585 - accuracy: 1.0000 -
    val_loss: 0.0627 - val_accuracy: 1.0000
Epoch 96/100
18/18 [==============================] - 0s 13ms/step - loss: 0.0601 - accuracy: 1.0000 -
    val_loss: 0.0610 - val_accuracy: 1.0000
Epoch 97/100
18/18 [==============================] - 0s 12ms/step - loss: 0.0570 - accuracy: 1.0000 -
    val_loss: 0.0609 - val_accuracy: 1.0000
Epoch 98/100
18/18 [==============================] - 0s 10ms/step - loss: 0.0579 - accuracy: 1.0000 -
    val_loss: 0.0590 - val_accuracy: 1.0000
Epoch 99/100
18/18 [==============================] - 0s 14ms/step - loss: 0.0547 - accuracy: 1.0000 -
    val_loss: 0.0574 - val_accuracy: 1.0000
Epoch 100/100
18/18 [==============================] - 0s 14ms/step - loss: 0.0556 - accuracy: 1.0000 -
    val_loss: 0.0559 - val_accuracy: 1.0000
24/24 [==============================] - 0s 5ms/step - loss: 0.0568 - accuracy: 1.0000
8/8 [==============================] - 0s 4ms/step - loss: 0.0595 - accuracy: 1.0000
```

```
results.append({"model": 'ConvNet', "train_score": train_score, "test_score": test_score})
```

```
results_df = pd.DataFrame(results)

plt.figure(figsize =(12,10));

plt.subplot(2,1,1)
sns.barplot(x=results_df.sort_values('test_score')['model'],
    y=results_df.sort_values('test_score')['train_score']);
plt.ylim(0,1);
plt.xlabel("")

plt.subplot(2,1,2)
sns.barplot(x=results_df.sort_values('test_score')['model'],
    y=results_df.sort_values('test_score')['test_score']);
plt.ylim(0,1);
```



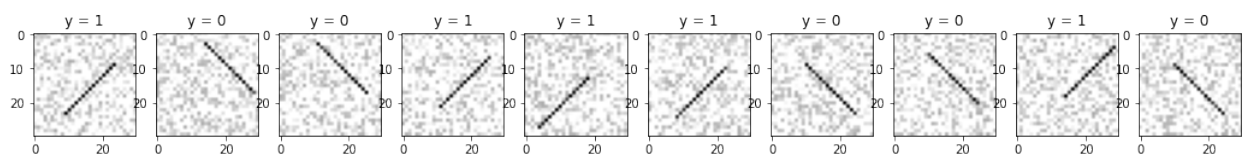
Using the same model on different slashes

Not only did our convolutional network learn forward and backward slashes - it can even generalize to slightly different forward and backward slashes.

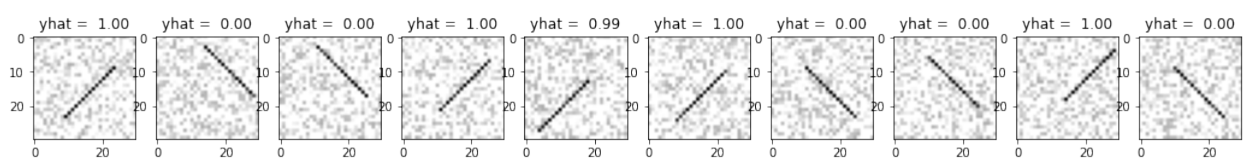
Let's generate data with heavier background noise, and longer slashes:

```
noise_scale = 0.65
s_pattern = 15
def gen_example_different(size=20, label=0):

    max_s_pattern = int(size // 4)
    pattern = 1- np.eye(s_pattern)
    if label:
        pattern = pattern[:, ::-1]
    ex = np.ones((size,size))
    point_loc = np.random.randint(0, size - s_pattern + 1,
                                  size=(2, ))  # random x,y point
    ex[point_loc[0]:point_loc[0] + s_pattern, point_loc[1]:point_loc[1] +
       s_pattern] = pattern  # set point to
    ex = ex + noise_scale*(np.random.rand(size, size) - .5)
```

```
    np.clip(ex,0.,1., out=ex)
    return ex

examples = []

n_side = 30
n_ex = 50 #number of examples in each class

for i in range(n_ex):
    examples.append(gen_example_different(size=n_side, label=0))
    examples.append(gen_example_different(size=n_side, label=1))

y_new = np.array([0,1]*n_ex)
x_new = np.stack(examples)

plt.figure(figsize=(18,4))

n_print = 10 # number of examples to show

ex_indices = np.random.choice(len(y_new), n_print, replace=False)
for i, index in enumerate(ex_indices):
    plt.subplot(1, n_print, i+1, )
    plt.imshow(x_new[index,...], cmap='gray')
    plt.title(f"y = {y_new[index]}")
```




```
plt.figure(figsize=(18,4))

for i, index in enumerate(ex_indices):
    plt.subplot(1, n_print, i+1, )
    plt.imshow(x_new[index,...], cmap='gray')
    plt.title("yhat =  %0.2f" % model_conv.predict(x_new[index].reshape((1,30,30,1))))
```



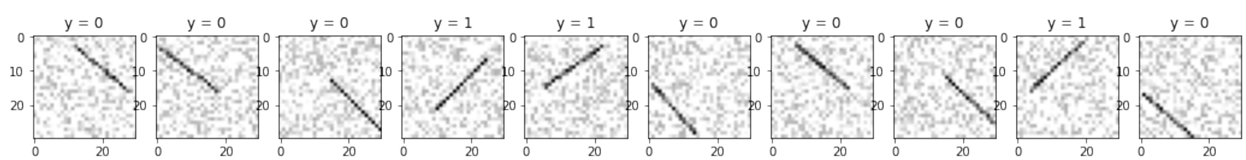

```
new_test_score = model_conv.evaluate(x_new[...,np.newaxis], y_new)[1]
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.0028 - accuracy: 1.0000
```

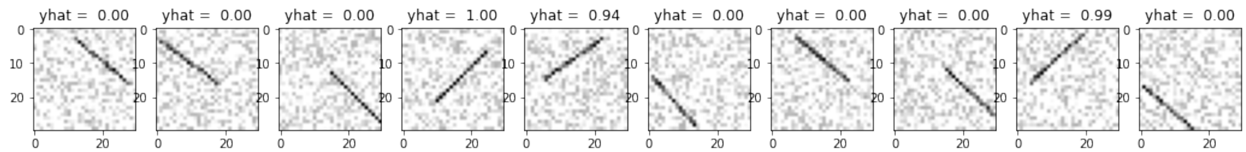
What about forward and backward slashes at different angles?

```
max_rot = 10
def gen_example_rotated(size=20, label=0):

    max_s_pattern = int(size // 4)
```

```
    s_pattern = 15
    pattern = 1- np.eye(s_pattern)
    if label:
        pattern = pattern[:, ::-1]
    ex = np.ones((size,size))
    point_loc = np.random.randint(0, size - s_pattern + 1,   size=(2, ))
    ex[point_loc[0]:point_loc[0] + s_pattern, point_loc[1]:point_loc[1] + s_pattern] =
        pattern
    rot_angle = np.random.uniform(-max_rot, max_rot)
    ex = scipy.ndimage.rotate(ex, angle=rot_angle, cval=1, reshape = False)
    ex = ex + noise_scale*(np.random.rand(size, size) - .5)

    np.clip(ex,0.,1., out=ex)
    return ex

examples = []

n_side = 30
n_ex = 50 #number of examples in each class

for i in range(n_ex):
    examples.append(gen_example_rotated(size=n_side, label=0))
    examples.append(gen_example_rotated(size=n_side, label=1))

y_new = np.array([0,1]*n_ex)
x_new = np.stack(examples)

plt.figure(figsize=(18,4))

n_print = 10 # number of examples to show

ex_indices = np.random.choice(len(y_new), n_print, replace=False)
for i, index in enumerate(ex_indices):
    plt.subplot(1, n_print, i+1, )
    plt.imshow(x_new[index,...], cmap='gray')
    plt.title(f"y = {y_new[index]}")
```




```
plt.figure(figsize=(18,4))

for i, index in enumerate(ex_indices):
    plt.subplot(1, n_print, i+1, )
    plt.imshow(x_new[index,...], cmap='gray')
    plt.title("yhat =  %0.2f" % model_conv.predict(x_new[index].reshape((1,30,30,1))))
```




```
new_test_score = model_conv.evaluate(x_new[...,np.newaxis], y_new)[1]
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.0225 - accuracy: 1.0000
```

Visualizing what the network learns

```
from ipywidgets import interactive
from ipywidgets import Layout
import ipywidgets as widgets

def plot_layer(test_idx, layer_idx):
  convout1_f = K.function(model_conv.inputs, [model_conv.layers[layer_idx].output])
  convolutions = np.squeeze(convout1_f(x[test_idx].reshape((1,30,30,1))))
  if (len(convolutions.shape)) > 1:
    m = convolutions.shape[2]
    n = int(np.ceil(np.sqrt(m)))

    # Visualization of each filter of the layer
    fig = plt.figure(figsize=(15,12))
    print(model_conv.layers[layer_idx].name)
    for i in range(m):
        ax = fig.add_subplot(n,n,i+1)
        ax.imshow(convolutions[:,:,i], cmap='gray')
        ax.set_title(i)
  else:
    print(model_conv.layers[layer_idx].name)
    plt.imshow(convolutions.reshape(1, convolutions.shape[0]), cmap='gray');
    plt.yticks([])
    plt.xticks(range(convolutions.shape[0]))

style = {'description_width': 'initial'}
layout = Layout(width="800px")
test_idx = widgets.IntSlider(min=0, max=len(x)-1, value=0, style=style, layout=layout)
layer_idx = widgets.IntSlider(min=0, max=len(model_conv.layers)-2, value=0, style=style,
    layout=layout)
interactive(plot_layer, test_idx=test_idx, layer_idx=layer_idx)
```

```
---------------------------------------------------------------------------
TraitError                                Traceback (most recent call last)
<ipython-input-26-79fd629fd80f> in <module>
     25 style = {'description_width': 'initial'}
     26 layout = Layout(width="800px")
---> 27 test_idx = widgets.IntSlider(min=0, max=len(x)-1, value=0, style=style,
    layout=layout)
     28 layer_idx = widgets.IntSlider(min=0, max=len(model_conv.layers)-2, value=0,
         style=style, layout=layout)
     29 interactive(plot_layer, test_idx=test_idx, layer_idx=layer_idx)
```

```
/usr/lib/python3/dist-packages/ipywidgets/widgets/widget_int.py in __init__(self, value,
    min, max, step, **kwargs)
     60         if step is not None:
     61             kwargs['step'] = step
---> 62         super(cls, self).__init__(**kwargs)
     63 
     64     __init__.__doc__ = _bounded_int_doc_t

/usr/lib/python3/dist-packages/ipywidgets/widgets/widget_int.py in __init__(self, value,
    min, max, step, **kwargs)
     97         if step is not None:
     98             kwargs['step'] = step
---> 99         super(_BoundedInt, self).__init__(**kwargs)
    100 
    101     @validate('value')

/usr/lib/python3/dist-packages/ipywidgets/widgets/widget_int.py in __init__(self, value,
    **kwargs)
     78         if value is not None:
     79             kwargs['value'] = value
---> 80         super(_Int, self).__init__(**kwargs)
     81 
     82 

/usr/lib/python3/dist-packages/ipywidgets/widgets/widget.py in __init__(self, **kwargs)
    198         """Public constructor"""
    199         self._model_id = kwargs.pop('model_id', None)
--> 200         super(Widget, self).__init__(**kwargs)
    201 
    202         Widget._call_widget_constructed(self)

/usr/lib/python3/dist-packages/traitlets/config/configurable.py in __init__(self, **kwargs)
     71 
     72         # load kwarg traits, other than config
---> 73         super(Configurable, self).__init__(**kwargs)
     74 
     75         # load config

/usr/lib/python3/dist-packages/traitlets/traitlets.py in __init__(self, *args, **kwargs)
    995             for key, value in kwargs.items():
    996                 if self.has_trait(key):
--> 997                     setattr(self, key, value)
    998                 else:
    999                     # passthrough args that don't set traits to super

/usr/lib/python3/dist-packages/traitlets/traitlets.py in __set__(self, obj, value)
    583             raise TraitError('The "%s" trait is read-only.' % self.name)
    584         else:
--> 585             self.set(obj, value)
    586 
    587     def _validate(self, obj, value):

/usr/lib/python3/dist-packages/traitlets/traitlets.py in set(self, obj, value)
```

```
    557
    558     def set(self, obj, value):
--> 559         new_value = self._validate(obj, value)
    560         try:
    561             old_value = obj._trait_values[self.name]

/usr/lib/python3/dist-packages/traitlets/traitlets.py in _validate(self, obj, value)
    589             return value
    590         if hasattr(self, 'validate'):
--> 591             value = self.validate(obj, value)
    592         if obj._cross_validation_lock is False:
    593             value = self._cross_validate(obj, value)

/usr/lib/python3/dist-packages/traitlets/traitlets.py in validate(self, obj, value)
   1675             return value
   1676         else:
-> 1677             self.error(obj, value)
   1678
   1679     def info(self):

/usr/lib/python3/dist-packages/traitlets/traitlets.py in error(self, obj, value)
   1522                 % (self.name, self.info(), msg)
   1523
-> 1524         raise TraitError(e)
   1525
   1526

TraitError: The 'style' trait of an IntSlider instance must be a SliderStyle, but a value of
    class 'dict' (i.e. {'description_width': 'initial'}) was specified.
```