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

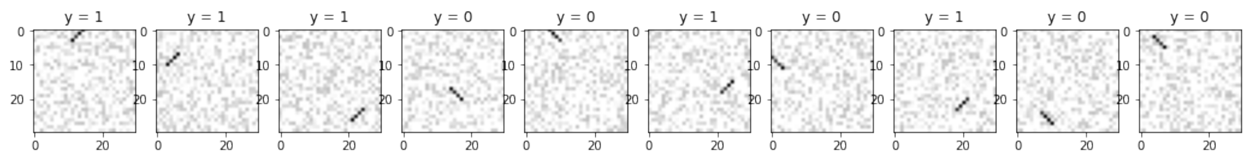
The "slash" dataset, developed by Sophie Searcy, is a set of images, each of which includes a "slash" on a background of random noise. The data is divided into two classes according to whether the slash is downward facing or upward facing.

```
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'l try to train some classic ML models on this dataset.

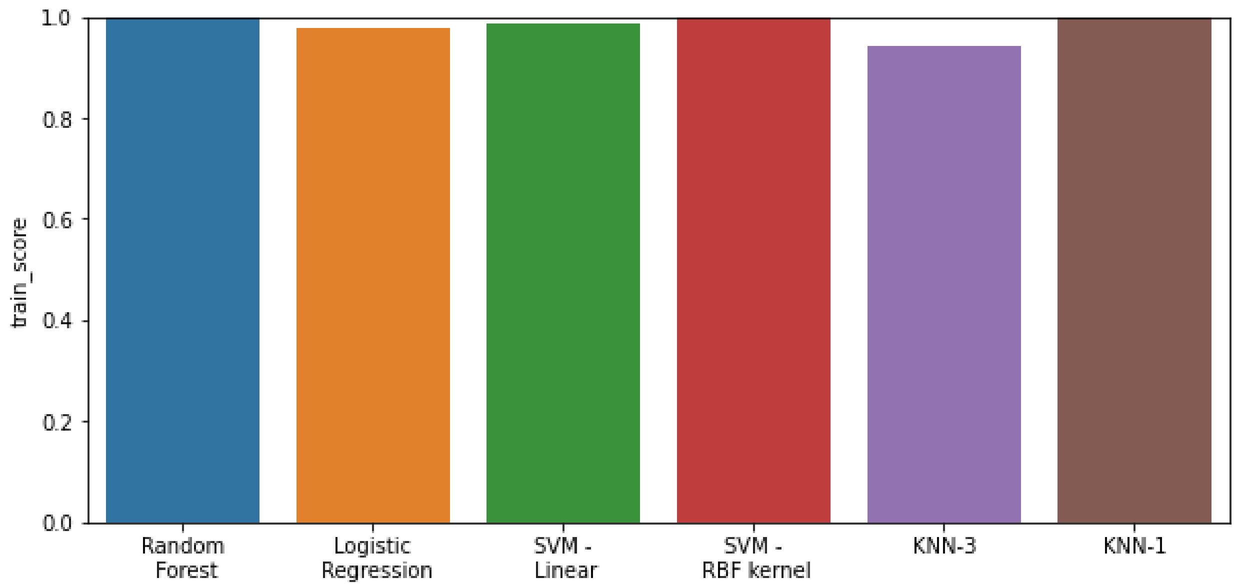
```
models = {
    "Logistic\n Regression": linear_model.LogisticRegression(),
    "KNN-1": neighbors.KNeighborsClassifier(n_neighbors=1),
    "KNN-3": neighbors.KNeighborsClassifier(n_neighbors=3),
```

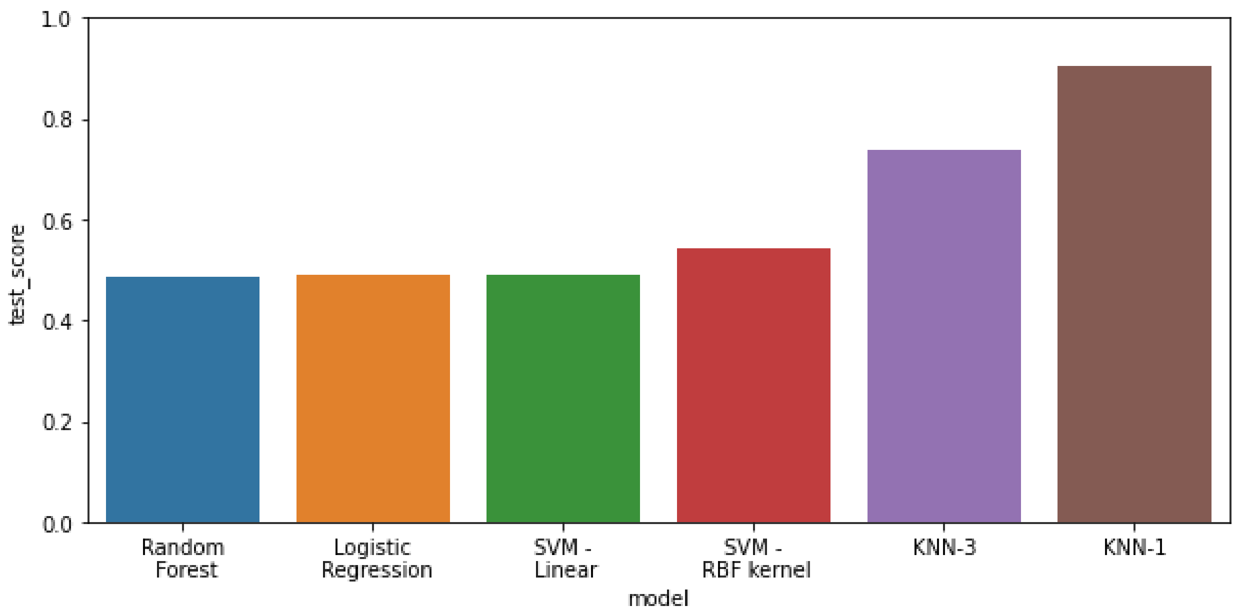
```
    "Random\n Forest": ensemble.RandomForestClassifier(n_estimators=100),
    "SVM -\n Linear": svm.SVC(kernel="linear"),
    "SVM -\n RBF kernel": svm.SVC(kernel="rbf")
}
```

```
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})
```

```
/usr/lib/python3/dist-packages/sklearn/linear_model/_logistic.py:938: ConvergenceWarning:
    lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
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 10ms/step - loss: 0.7058 - accuracy: 0.5196 -
    val_loss: 0.6991 - val_accuracy: 0.5266
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7090 - accuracy: 0.4733 -
    val_loss: 0.7063 - val_accuracy: 0.4734
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6958 - accuracy: 0.4911 -
    val_loss: 0.7054 - val_accuracy: 0.4734

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 4/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6960 - accuracy: 0.4858 -
    val_loss: 0.6921 - val_accuracy: 0.5266
Epoch 5/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6961 - accuracy: 0.5053 -
    val_loss: 0.7001 - val_accuracy: 0.4734
Epoch 6/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6974 - accuracy: 0.4609 -
    val_loss: 0.6947 - val_accuracy: 0.4734

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
```

```
Epoch 7/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6933 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734
Epoch 8/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6928 - accuracy: 0.5125 -
    val_loss: 0.6930 - val_accuracy: 0.5053

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 9/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6929 - accuracy: 0.4893 -
    val_loss: 0.6926 - val_accuracy: 0.5319
Epoch 10/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6924 - accuracy: 0.5196 -
    val_loss: 0.6950 - val_accuracy: 0.4734

Epoch 00010: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
Epoch 11/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6927 - accuracy: 0.5089 -
    val_loss: 0.6945 - val_accuracy: 0.4734
Epoch 12/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6927 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734

Epoch 00012: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
Epoch 13/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6925 - accuracy: 0.5089 -
    val_loss: 0.6940 - val_accuracy: 0.4734
Epoch 14/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6924 - accuracy: 0.5089 -
    val_loss: 0.6938 - val_accuracy: 0.4734

Epoch 00014: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
Epoch 15/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6923 - accuracy: 0.5089 -
    val_loss: 0.6938 - val_accuracy: 0.4734
Epoch 16/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6923 - accuracy: 0.5089 -
    val_loss: 0.6938 - val_accuracy: 0.4734

Epoch 00016: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
Epoch 17/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6923 - accuracy: 0.5089 -
    val_loss: 0.6938 - val_accuracy: 0.4734
Epoch 18/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6923 - accuracy: 0.5089 -
    val_loss: 0.6940 - val_accuracy: 0.4734

Epoch 00018: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.
Epoch 19/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6923 - accuracy: 0.5089 -
    val_loss: 0.6940 - val_accuracy: 0.4734
Epoch 20/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6922 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734
```

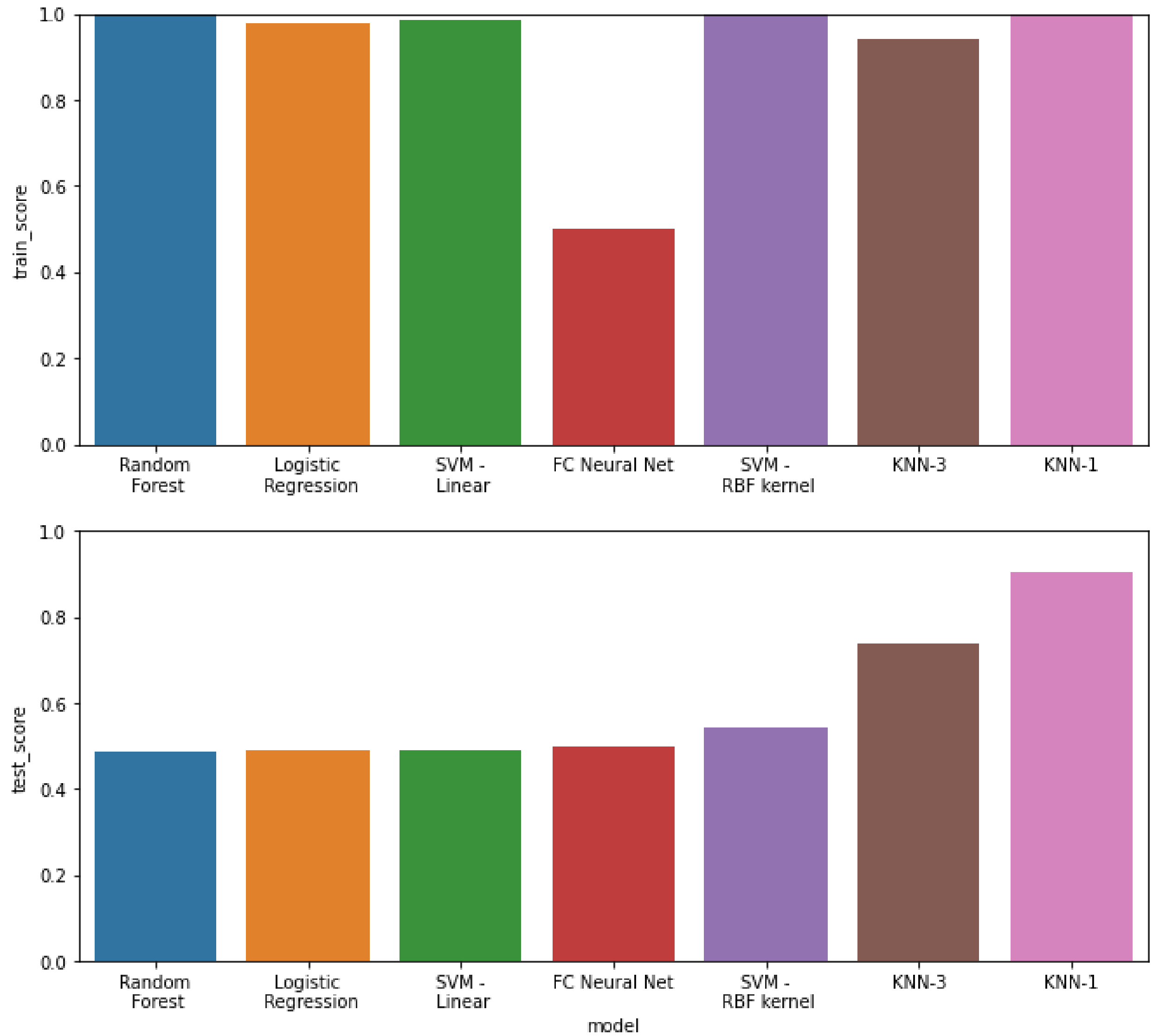
```
Epoch 00020: ReduceLROnPlateau reducing learning rate to 1.9531250927684596e-06.
Epoch 21/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6922 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734
Epoch 22/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6922 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734

Epoch 00022: ReduceLROnPlateau reducing learning rate to 9.765625463842298e-07.
Epoch 23/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6922 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734
Epoch 24/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6922 - accuracy: 0.5089 -
    val_loss: 0.6939 - val_accuracy: 0.4734

Epoch 00024: ReduceLROnPlateau reducing learning rate to 4.882812731921149e-07.
Restoring model weights from the end of the best epoch.
Epoch 00024: 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: 0.6931 - accuracy: 0.5000
8/8 [==============================] - 0s 908us/step - loss: 0.6931 - 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"))

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
_________________________________________________________________
```

```
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 12ms/step - loss: 0.6873 - accuracy: 0.5107 -
    val_loss: 0.6918 - val_accuracy: 0.5266
Epoch 2/100
18/18 [==============================] - 0s 7ms/step - loss: 0.6765 - accuracy: 0.8577 -
    val_loss: 0.6916 - val_accuracy: 0.5266
Epoch 3/100
18/18 [==============================] - 0s 8ms/step - loss: 0.6639 - accuracy: 0.9413 -
    val_loss: 0.6912 - val_accuracy: 0.9628
Epoch 4/100
18/18 [==============================] - 0s 7ms/step - loss: 0.6473 - accuracy: 0.9715 -
    val_loss: 0.6901 - val_accuracy: 0.4734
Epoch 5/100
18/18 [==============================] - 0s 7ms/step - loss: 0.6244 - accuracy: 0.9947 -
    val_loss: 0.6877 - val_accuracy: 0.4734
Epoch 6/100
18/18 [==============================] - 0s 7ms/step - loss: 0.5953 - accuracy: 0.9911 -
    val_loss: 0.6819 - val_accuracy: 0.9096
Epoch 7/100
18/18 [==============================] - 0s 8ms/step - loss: 0.5566 - accuracy: 0.9947 -
    val_loss: 0.6750 - val_accuracy: 0.9628
Epoch 8/100
```

```
18/18 [==============================] - 0s 7ms/step - loss: 0.5061 - accuracy: 0.9858 -
    val_loss: 0.6731 - val_accuracy: 0.4734
Epoch 9/100
18/18 [==============================] - 0s 8ms/step - loss: 0.4468 - accuracy: 1.0000 -
    val_loss: 0.7511 - val_accuracy: 0.4734
Epoch 10/100
18/18 [==============================] - 0s 7ms/step - loss: 0.3783 - accuracy: 0.9982 -
    val_loss: 0.8749 - val_accuracy: 0.4734

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.000500000237487257.
Epoch 11/100
18/18 [==============================] - 0s 7ms/step - loss: 0.3324 - accuracy: 0.9982 -
    val_loss: 0.9042 - val_accuracy: 0.4734
Epoch 12/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2963 - accuracy: 0.9982 -
    val_loss: 0.9495 - val_accuracy: 0.4734

Epoch 00012: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 13/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2693 - accuracy: 1.0000 -
    val_loss: 0.9276 - val_accuracy: 0.4734
Epoch 14/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2550 - accuracy: 1.0000 -
    val_loss: 0.9100 - val_accuracy: 0.4734

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 15/100
18/18 [==============================] - 0s 7ms/step - loss: 0.2446 - accuracy: 1.0000 -
    val_loss: 0.8956 - val_accuracy: 0.4734
Epoch 16/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2407 - accuracy: 1.0000 -
    val_loss: 0.8715 - val_accuracy: 0.4734

Epoch 00016: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
Epoch 17/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2350 - accuracy: 1.0000 -
    val_loss: 0.8325 - val_accuracy: 0.4734
Epoch 18/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2307 - accuracy: 1.0000 -
    val_loss: 0.7972 - val_accuracy: 0.4734

Epoch 00018: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
Epoch 19/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2285 - accuracy: 1.0000 -
    val_loss: 0.7574 - val_accuracy: 0.4734
Epoch 20/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2254 - accuracy: 1.0000 -
    val_loss: 0.7200 - val_accuracy: 0.4734

Epoch 00020: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
Epoch 21/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2255 - accuracy: 1.0000 -
    val_loss: 0.6787 - val_accuracy: 0.4734
Epoch 22/100
```

```
18/18 [==============================] - 0s 9ms/step - loss: 0.2234 - accuracy: 1.0000 -
    val_loss: 0.6425 - val_accuracy: 0.4734
Epoch 23/100
18/18 [==============================] - 0s 7ms/step - loss: 0.2215 - accuracy: 1.0000 -
    val_loss: 0.6048 - val_accuracy: 0.4734
Epoch 24/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2204 - accuracy: 1.0000 -
    val_loss: 0.5676 - val_accuracy: 0.4787
Epoch 25/100
18/18 [==============================] - 0s 16ms/step - loss: 0.2185 - accuracy: 1.0000 -
    val_loss: 0.5350 - val_accuracy: 0.5000
Epoch 26/100
18/18 [==============================] - 0s 15ms/step - loss: 0.2213 - accuracy: 1.0000 -
    val_loss: 0.5021 - val_accuracy: 0.5904
Epoch 27/100
18/18 [==============================] - 0s 14ms/step - loss: 0.2215 - accuracy: 1.0000 -
    val_loss: 0.4724 - val_accuracy: 0.6915
Epoch 28/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2166 - accuracy: 1.0000 -
    val_loss: 0.4454 - val_accuracy: 0.7713
Epoch 29/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2174 - accuracy: 1.0000 -
    val_loss: 0.4187 - val_accuracy: 0.8670
Epoch 30/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2197 - accuracy: 1.0000 -
    val_loss: 0.3943 - val_accuracy: 0.9149
Epoch 31/100
18/18 [==============================] - 0s 9ms/step - loss: 0.2162 - accuracy: 1.0000 -
    val_loss: 0.3735 - val_accuracy: 0.9362
Epoch 32/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2159 - accuracy: 1.0000 -
    val_loss: 0.3539 - val_accuracy: 0.9574
Epoch 33/100
18/18 [==============================] - 0s 14ms/step - loss: 0.2153 - accuracy: 1.0000 -
    val_loss: 0.3371 - val_accuracy: 0.9574
Epoch 34/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2163 - accuracy: 1.0000 -
    val_loss: 0.3228 - val_accuracy: 0.9681
Epoch 35/100
18/18 [==============================] - 0s 14ms/step - loss: 0.2124 - accuracy: 1.0000 -
    val_loss: 0.3094 - val_accuracy: 0.9681
Epoch 36/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2128 - accuracy: 1.0000 -
    val_loss: 0.2977 - val_accuracy: 0.9894
Epoch 37/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2110 - accuracy: 1.0000 -
    val_loss: 0.2875 - val_accuracy: 0.9894
Epoch 38/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2126 - accuracy: 1.0000 -
    val_loss: 0.2800 - val_accuracy: 0.9894
Epoch 39/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2160 - accuracy: 1.0000 -
    val_loss: 0.2732 - val_accuracy: 0.9947
Epoch 40/100
```

```
18/18 [==============================] - 0s 8ms/step - loss: 0.2094 - accuracy: 1.0000 -
    val_loss: 0.2703 - val_accuracy: 0.9894
Epoch 41/100
18/18 [==============================] - 0s 7ms/step - loss: 0.2128 - accuracy: 1.0000 -
    val_loss: 0.2670 - val_accuracy: 0.9894
Epoch 42/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2102 - accuracy: 1.0000 -
    val_loss: 0.2625 - val_accuracy: 0.9894
Epoch 43/100
18/18 [==============================] - 0s 13ms/step - loss: 0.2089 - accuracy: 1.0000 -
    val_loss: 0.2617 - val_accuracy: 0.9894
Epoch 44/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2046 - accuracy: 1.0000 -
    val_loss: 0.2618 - val_accuracy: 0.9894
Epoch 45/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2023 - accuracy: 1.0000 -
    val_loss: 0.2653 - val_accuracy: 0.9681

Epoch 00045: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
Epoch 46/100
18/18 [==============================] - 0s 14ms/step - loss: 0.2030 - accuracy: 1.0000 -
    val_loss: 0.2563 - val_accuracy: 0.9894
Epoch 47/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2084 - accuracy: 1.0000 -
    val_loss: 0.2479 - val_accuracy: 1.0000
Epoch 48/100
18/18 [==============================] - 0s 10ms/step - loss: 0.2031 - accuracy: 1.0000 -
    val_loss: 0.2405 - val_accuracy: 1.0000
Epoch 49/100
18/18 [==============================] - 0s 7ms/step - loss: 0.2029 - accuracy: 0.9982 -
    val_loss: 0.2339 - val_accuracy: 1.0000
Epoch 50/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2034 - accuracy: 1.0000 -
    val_loss: 0.2291 - val_accuracy: 1.0000
Epoch 51/100
18/18 [==============================] - 0s 8ms/step - loss: 0.2027 - accuracy: 1.0000 -
    val_loss: 0.2240 - val_accuracy: 1.0000
Epoch 52/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1968 - accuracy: 1.0000 -
    val_loss: 0.2198 - val_accuracy: 1.0000
Epoch 53/100
18/18 [==============================] - 0s 11ms/step - loss: 0.1979 - accuracy: 1.0000 -
    val_loss: 0.2159 - val_accuracy: 1.0000
Epoch 54/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1973 - accuracy: 1.0000 -
    val_loss: 0.2125 - val_accuracy: 1.0000
Epoch 55/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1964 - accuracy: 1.0000 -
    val_loss: 0.2092 - val_accuracy: 1.0000
Epoch 56/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1975 - accuracy: 1.0000 -
    val_loss: 0.2068 - val_accuracy: 1.0000
Epoch 57/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1963 - accuracy: 1.0000 -
```

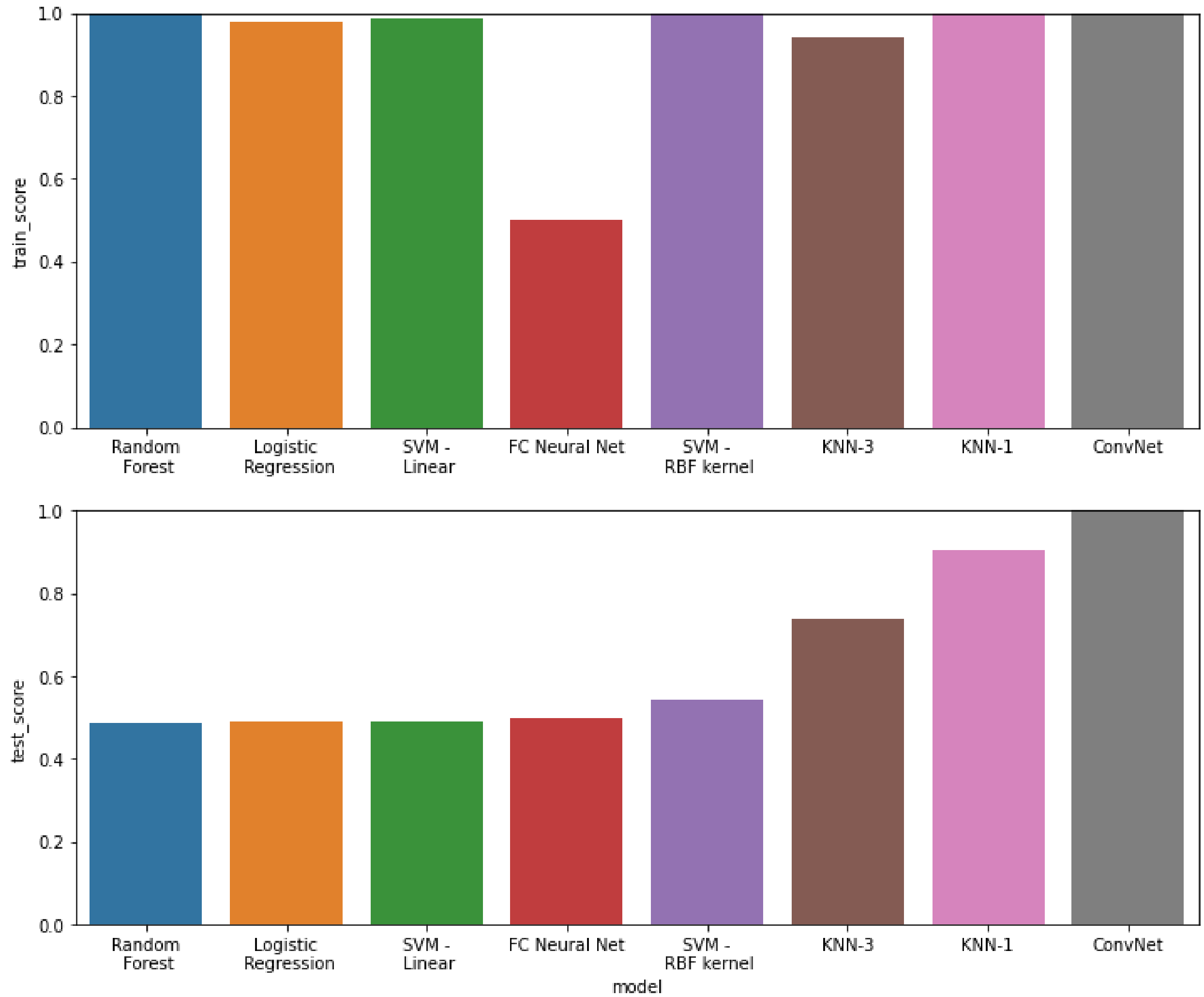
```
    val_loss: 0.2042 - val_accuracy: 1.0000
Epoch 58/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1951 - accuracy: 1.0000 -
    val_loss: 0.2019 - val_accuracy: 1.0000
Epoch 59/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1985 - accuracy: 1.0000 -
    val_loss: 0.1997 - val_accuracy: 1.0000
Epoch 60/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1942 - accuracy: 1.0000 -
    val_loss: 0.1979 - val_accuracy: 1.0000
Epoch 61/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1887 - accuracy: 1.0000 -
    val_loss: 0.1961 - val_accuracy: 1.0000
Epoch 62/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1938 - accuracy: 1.0000 -
    val_loss: 0.1944 - val_accuracy: 1.0000
Epoch 63/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1903 - accuracy: 1.0000 -
    val_loss: 0.1927 - val_accuracy: 1.0000
Epoch 64/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1926 - accuracy: 1.0000 -
    val_loss: 0.1914 - val_accuracy: 1.0000
Epoch 65/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1897 - accuracy: 1.0000 -
    val_loss: 0.1907 - val_accuracy: 1.0000
Epoch 66/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1891 - accuracy: 1.0000 -
    val_loss: 0.1897 - val_accuracy: 1.0000
Epoch 67/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1911 - accuracy: 1.0000 -
    val_loss: 0.1888 - val_accuracy: 1.0000
Epoch 68/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1897 - accuracy: 1.0000 -
    val_loss: 0.1878 - val_accuracy: 1.0000
Epoch 69/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1885 - accuracy: 1.0000 -
    val_loss: 0.1872 - val_accuracy: 1.0000
Epoch 70/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1881 - accuracy: 1.0000 -
    val_loss: 0.1863 - val_accuracy: 1.0000
Epoch 71/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1914 - accuracy: 1.0000 -
    val_loss: 0.1854 - val_accuracy: 1.0000
Epoch 72/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1864 - accuracy: 1.0000 -
    val_loss: 0.1849 - val_accuracy: 1.0000
Epoch 73/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1885 - accuracy: 1.0000 -
    val_loss: 0.1840 - val_accuracy: 1.0000
Epoch 74/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1846 - accuracy: 1.0000 -
    val_loss: 0.1835 - val_accuracy: 1.0000
Epoch 75/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1888 - accuracy: 1.0000 -
```

```
    val_loss: 0.1828 - val_accuracy: 1.0000
Epoch 76/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1876 - accuracy: 1.0000 -
    val_loss: 0.1822 - val_accuracy: 1.0000
Epoch 77/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1843 - accuracy: 1.0000 -
    val_loss: 0.1816 - val_accuracy: 1.0000
Epoch 78/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1869 - accuracy: 1.0000 -
    val_loss: 0.1813 - val_accuracy: 1.0000
Epoch 79/100
18/18 [==============================] - 0s 17ms/step - loss: 0.1839 - accuracy: 1.0000 -
    val_loss: 0.1807 - val_accuracy: 1.0000
Epoch 80/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1845 - accuracy: 1.0000 -
    val_loss: 0.1802 - val_accuracy: 1.0000
Epoch 81/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1876 - accuracy: 1.0000 -
    val_loss: 0.1797 - val_accuracy: 1.0000
Epoch 82/100
18/18 [==============================] - 0s 17ms/step - loss: 0.1816 - accuracy: 1.0000 -
    val_loss: 0.1793 - val_accuracy: 1.0000
Epoch 83/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1799 - accuracy: 1.0000 -
    val_loss: 0.1789 - val_accuracy: 1.0000
Epoch 84/100
18/18 [==============================] - 0s 15ms/step - loss: 0.1837 - accuracy: 1.0000 -
    val_loss: 0.1783 - val_accuracy: 1.0000
Epoch 85/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1823 - accuracy: 1.0000 -
    val_loss: 0.1779 - val_accuracy: 1.0000
Epoch 86/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1836 - accuracy: 1.0000 -
    val_loss: 0.1774 - val_accuracy: 1.0000
Epoch 87/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1836 - accuracy: 1.0000 -
    val_loss: 0.1766 - val_accuracy: 1.0000
Epoch 88/100
18/18 [==============================] - 0s 11ms/step - loss: 0.1853 - accuracy: 1.0000 -
    val_loss: 0.1761 - val_accuracy: 1.0000
Epoch 89/100
18/18 [==============================] - 0s 9ms/step - loss: 0.1778 - accuracy: 1.0000 -
    val_loss: 0.1758 - val_accuracy: 1.0000
Epoch 90/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1799 - accuracy: 1.0000 -
    val_loss: 0.1755 - val_accuracy: 1.0000
Epoch 91/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1842 - accuracy: 1.0000 -
    val_loss: 0.1750 - val_accuracy: 1.0000
Epoch 92/100
18/18 [==============================] - 0s 10ms/step - loss: 0.1860 - accuracy: 1.0000 -
    val_loss: 0.1746 - val_accuracy: 1.0000
Epoch 93/100
18/18 [==============================] - 0s 8ms/step - loss: 0.1791 - accuracy: 1.0000 -
```

```
    val_loss: 0.1741 - val_accuracy: 1.0000
Epoch 94/100
18/18 [==============================] - 0s 14ms/step - loss: 0.1803 - accuracy: 1.0000 -
    val_loss: 0.1738 - val_accuracy: 1.0000
Epoch 95/100
18/18 [==============================] - 0s 13ms/step - loss: 0.1780 - accuracy: 1.0000 -
    val_loss: 0.1736 - val_accuracy: 1.0000
Epoch 96/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1814 - accuracy: 1.0000 -
    val_loss: 0.1731 - val_accuracy: 1.0000
Epoch 97/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1788 - accuracy: 1.0000 -
    val_loss: 0.1728 - val_accuracy: 1.0000
Epoch 98/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1784 - accuracy: 1.0000 -
    val_loss: 0.1723 - val_accuracy: 1.0000
Epoch 99/100
18/18 [==============================] - 0s 7ms/step - loss: 0.1761 - accuracy: 1.0000 -
    val_loss: 0.1721 - val_accuracy: 1.0000
Epoch 100/100
18/18 [==============================] - 0s 12ms/step - loss: 0.1795 - accuracy: 1.0000 -
    val_loss: 0.1714 - val_accuracy: 1.0000
24/24 [==============================] - 0s 4ms/step - loss: 0.1720 - accuracy: 1.0000
8/8 [==============================] - 0s 3ms/step - loss: 0.1817 - 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);
```



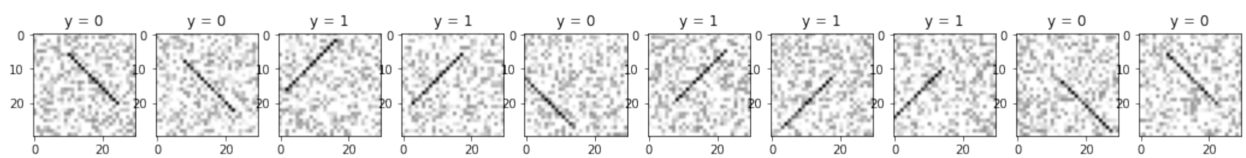
Using the same model on different slashes

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

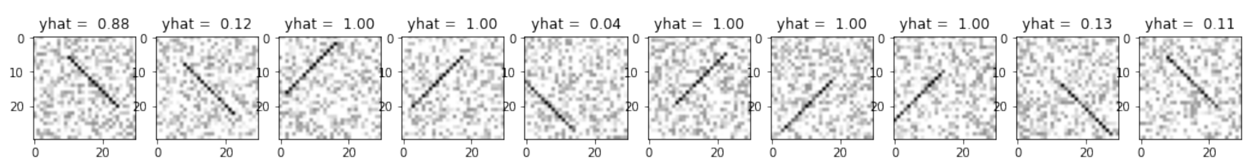
Let's generate data with heavier background noise, and longer slashes:

```
noise_scale = 0.9
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))))
```



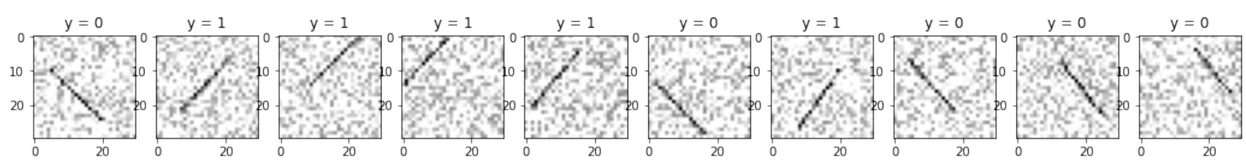

```
new_test_score = model_conv.evaluate(x_new[...,np.newaxis], y_new)[1]
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.1468 - accuracy: 0.9400
```

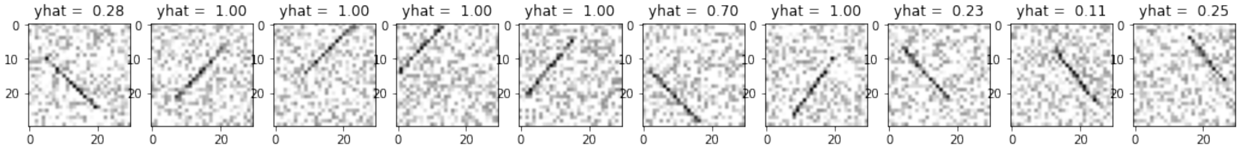
What about forward and backward slashes at different angles?

```
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: 0.1569 - accuracy: 0.9600
```

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.
```