```
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Flatten
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import pickle
import pandas as pd
import numpy as np
import random
```

```
!git clone https://bitbucket.org/jadslim/german-traffic-signs
```

```
Cloning into 'german-traffic-signs'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 117.80 MiB | 5.29 MiB/s, done.
Updating files: 100% (4/4), done.
```

```
!ls german-traffic-signs
```

```
signnames.csv  test.p  train.p  valid.p
```

```
df = pd.read_csv("german-traffic-signs/signnames.csv")
```

```
df.head()
```

	ClassId	SignName
0	0	Speed limit (20km/h)
1	1	Speed limit (30km/h)
2	2	Speed limit (50km/h)
3	3	Speed limit (60km/h)
4	4	Speed limit (70km/h)

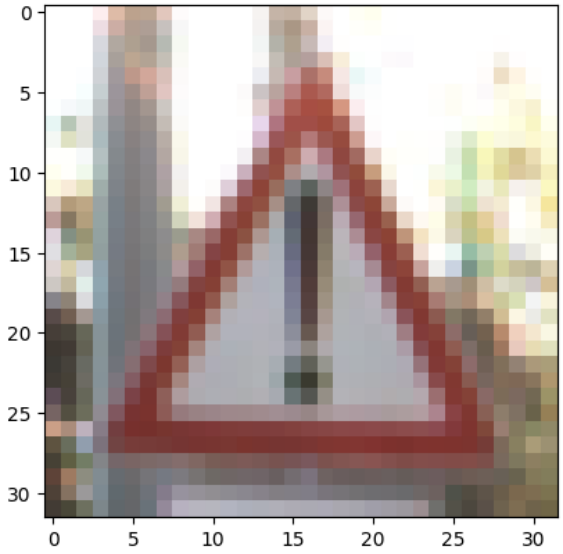
Next steps: Generate code with `df` View recommended plots

```
with open("german-traffic-signs/train.p", mode = "rb") as training:
    train = pickle.load(training)

with open("german-traffic-signs/valid.p", mode = "rb") as validation:
    valid = pickle.load(validation)

with open("german-traffic-signs/test.p", mode = "rb") as test:
    test = pickle.load(test)


X_train, y_train = train["features"], train["labels"]
X_validation, y_validation = valid["features"], valid["labels"]
X_test, y_test = test["features"], test["labels"]


print(X_train.shape)
print(X_validation.shape)
print(X_test.shape)
```

```
(34799, 32, 32, 3)
(4410, 32, 32, 3)
(12630, 32, 32, 3)
```

```
index = np.random.randint(1, len(X_train))
plt.imshow(X_train[index])
print("Image Label: = {}".format(y_train[index]))
```

```
Image Label: = 18
```

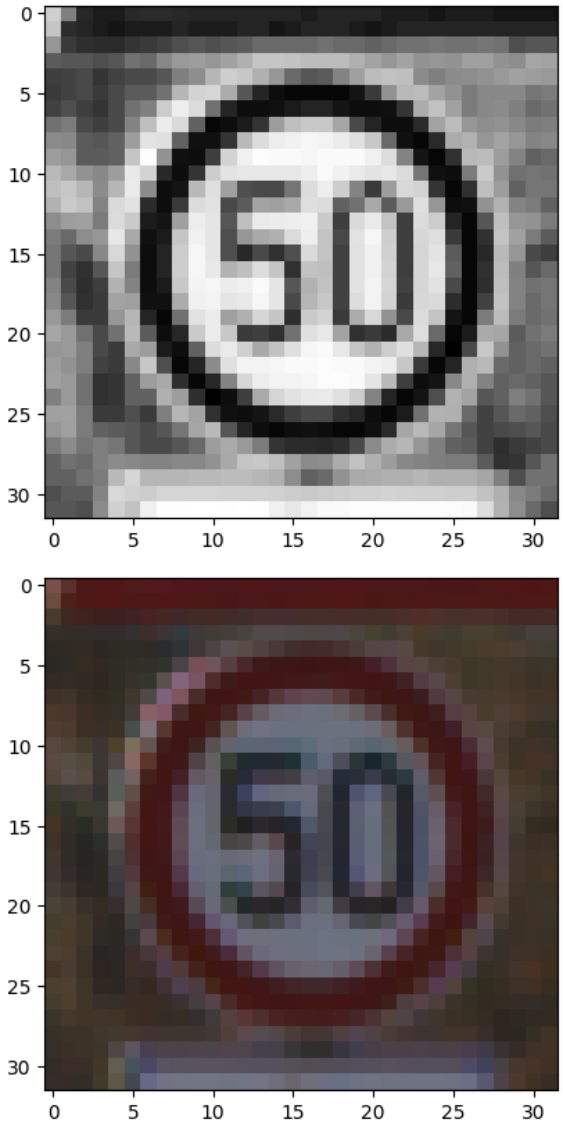


```
#Convert images to Grayscale
#Histogram Equalization
#Normalization

#Shuffling Images
from sklearn.utils import shuffle
X_train, y_train = shuffle(X_train, y_train)


def preprocessing(img):
    #Convert to Grayscale Images
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #Applyting Histogram Equalization
    img = cv2.equalizeHist(img)
    #Normalization
    img = img/255
    return img


X_train_processed = np.array(list(map(preprocessing, X_train)))
X_validation_processed = np.array(list(map(preprocessing, X_validation)))
X_test_processed = np.array(list(map(preprocessing, X_test)))


#Reshape X_train, X_test, X_validation

X_train_processed = X_train_processed.reshape(X_train_processed.shape[0], X_train_processed.shape[1], X_train_processed.shape[2], 1)
X_test_processed = X_test_processed.reshape(X_test_processed.shape[0], X_test_processed.shape[1], X_test_processed.shape[2], 1)
X_validation_processed = X_validation_processed.reshape(X_validation_processed.shape[0], X_validation_processed.shape[1], X_validation_processed.shape[2], 1)



print(X_train_processed.shape)
print(X_test_processed.shape)
print(X_validation_processed.shape)
```

```
(34799, 32, 32, 1)
(12630, 32, 32, 1)
(4410, 32, 32, 1)
```

```
i = random.randint(1, len(X_train))
plt.imshow(X_train_processed[i].squeeze(), cmap = "gray")
plt.figure()
plt.imshow(X_train[i].squeeze())
```

```
<matplotlib.image.AxesImage at 0x7fc0c26a48b0>
```



```
model = Sequential()

model.add(Conv2D(filters = 32,
                 kernel_size = (5,5),
                 activation = "relu",
                 input_shape = X_train_processed.shape[1:]))

model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25)) #To Reduce Overfitting of the data

model.add(Conv2D(filters = 64,
                 kernel_size = (3,3),
                 activation = "relu"))

model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))

model.add(Dense(43, activation = "softmax"))


model.summary()
```

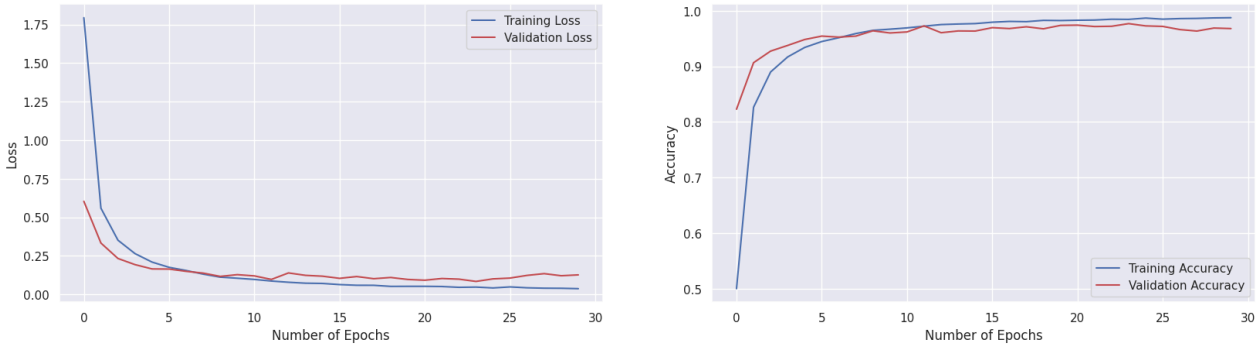
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        832

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 dropout (Dropout)           (None, 14, 14, 32)        0

 conv2d_1 (Conv2D)           (None, 12, 12, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 6, 6, 64)          0

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 256)               590080

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 620459 (2.37 MB)
Trainable params: 620459 (2.37 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer = Adam(learning_rate=0.001),
              loss = "sparse_categorical_crossentropy",
              metrics = ["accuracy"])


history = model.fit(X_train_processed, y_train,
          batch_size = 128, #How many images will be feeding at once
          epochs = 30,
          validation_data = (X_validation_processed, y_validation))
```

```
Epoch 2/30
272/272 [==============================] - 2s 6ms/step - loss: 0.5591 - accuracy: 0.8264 - val_loss: 0.3328 - val_accuracy: 0.9066
Epoch 3/30
272/272 [==============================] - 2s 6ms/step - loss: 0.3512 - accuracy: 0.8899 - val_loss: 0.2326 - val_accuracy: 0.9274
Epoch 4/30
272/272 [==============================] - 2s 6ms/step - loss: 0.2642 - accuracy: 0.9169 - val_loss: 0.1927 - val_accuracy: 0.9376
Epoch 5/30
272/272 [==============================] - 2s 6ms/step - loss: 0.2091 - accuracy: 0.9340 - val_loss: 0.1648 - val_accuracy: 0.9483
Epoch 6/30
272/272 [==============================] - 2s 6ms/step - loss: 0.1751 - accuracy: 0.9447 - val_loss: 0.1638 - val_accuracy: 0.9544
Epoch 7/30
272/272 [==============================] - 2s 6ms/step - loss: 0.1546 - accuracy: 0.9515 - val_loss: 0.1491 - val_accuracy: 0.9526
Epoch 8/30
272/272 [==============================] - 2s 7ms/step - loss: 0.1310 - accuracy: 0.9590 - val_loss: 0.1377 - val_accuracy: 0.9544
Epoch 9/30
272/272 [==============================] - 2s 7ms/step - loss: 0.1114 - accuracy: 0.9648 - val_loss: 0.1164 - val_accuracy: 0.9639
Epoch 10/30
272/272 [==============================] - 2s 6ms/step - loss: 0.1045 - accuracy: 0.9669 - val_loss: 0.1277 - val_accuracy: 0.9601
Epoch 11/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0966 - accuracy: 0.9692 - val_loss: 0.1195 - val_accuracy: 0.9619
Epoch 12/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0862 - accuracy: 0.9723 - val_loss: 0.0967 - val_accuracy: 0.9728
Epoch 13/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0783 - accuracy: 0.9752 - val_loss: 0.1388 - val_accuracy: 0.9605
Epoch 14/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0722 - accuracy: 0.9762 - val_loss: 0.1232 - val_accuracy: 0.9637
Epoch 15/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0706 - accuracy: 0.9769 - val_loss: 0.1176 - val_accuracy: 0.9635
Epoch 16/30
272/272 [==============================] - 2s 8ms/step - loss: 0.0631 - accuracy: 0.9794 - val_loss: 0.1037 - val_accuracy: 0.9696
Epoch 17/30
272/272 [==============================] - 2s 7ms/step - loss: 0.0589 - accuracy: 0.9809 - val_loss: 0.1154 - val_accuracy: 0.9680
Epoch 18/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0587 - accuracy: 0.9805 - val_loss: 0.1015 - val_accuracy: 0.9712
Epoch 19/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0515 - accuracy: 0.9827 - val_loss: 0.1092 - val_accuracy: 0.9676
Epoch 20/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0518 - accuracy: 0.9824 - val_loss: 0.0965 - val_accuracy: 0.9737
Epoch 21/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0519 - accuracy: 0.9830 - val_loss: 0.0915 - val_accuracy: 0.9741
Epoch 22/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0507 - accuracy: 0.9834 - val_loss: 0.1027 - val_accuracy: 0.9717
Epoch 23/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0454 - accuracy: 0.9848 - val_loss: 0.0984 - val_accuracy: 0.9721
Epoch 24/30
272/272 [==============================] - 2s 7ms/step - loss: 0.0470 - accuracy: 0.9845 - val_loss: 0.0837 - val_accuracy: 0.9769
Epoch 25/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0413 - accuracy: 0.9869 - val_loss: 0.1003 - val_accuracy: 0.9728
Epoch 26/30
```

```
272/272 [==============================] - 2s 6ms/step - loss: 0.0423 - accuracy: 0.9859 - val_loss: 0.1230 - val_accuracy: 0.9662
Epoch 28/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0398 - accuracy: 0.9863 - val_loss: 0.1342 - val_accuracy: 0.9635
Epoch 29/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0392 - accuracy: 0.9872 - val_loss: 0.1207 - val_accuracy: 0.9689
Epoch 30/30
272/272 [==============================] - 2s 6ms/step - loss: 0.0368 - accuracy: 0.9875 - val_loss: 0.1262 - val_accuracy: 0.9680
```

```
#Create loss and accuracy plot

import matplotlib.pyplot as plt
sns.set()
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], color='b', label="Training Loss")
plt.plot(history.history['val_loss'], color='r', label="Validation Loss")
plt.legend()
plt.xlabel("Number of Epochs")
plt.ylabel("Loss")

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], color='b', label="Training Accuracy")
plt.plot(history.history['val_accuracy'], color='r', label="Validation Accuracy")
plt.legend()
plt.xlabel("Number of Epochs")
plt.ylabel("Accuracy")
plt.show()
```



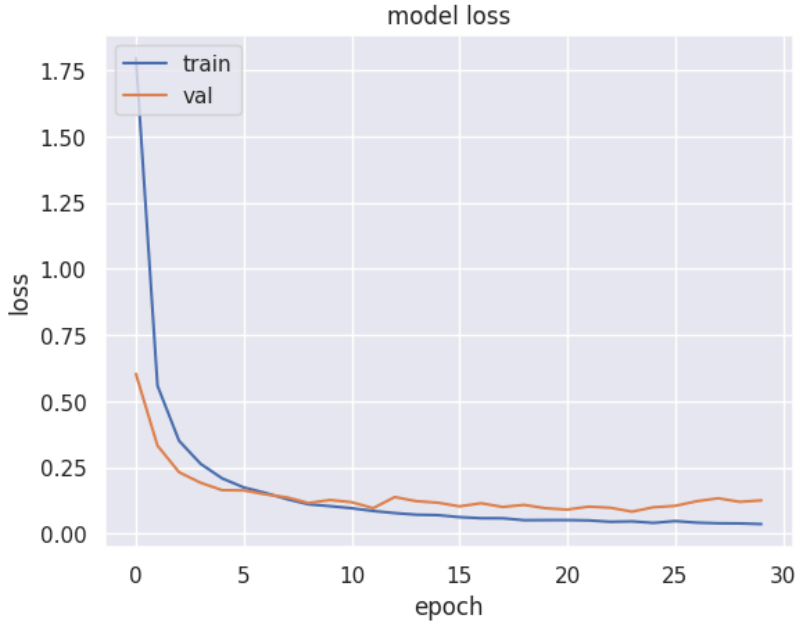

```
score = model.evaluate(X_test_processed, y_test, verbose = 0)
print("Test Loss: ", score[0])
print("Test Accuracy: ", score[1])
```

```
Test Loss:  0.1725344955921173
Test Accuracy:  0.9600158333778381
```

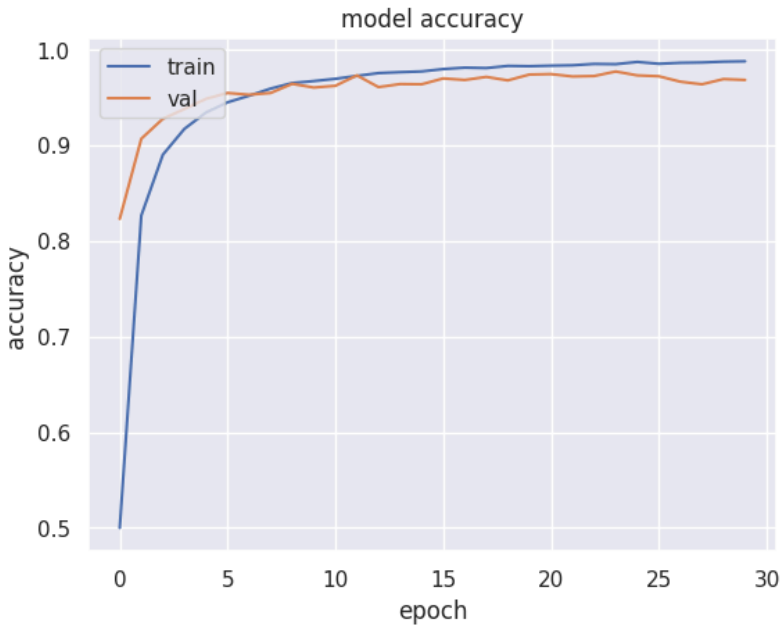
```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
prediction = model.predict(X_test_processed)
```

```
395/395 [==============================] - 1s 3ms/step
```

```
#Generate confusion matrix

import numpy as np
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Get predictions from the model
predictions = model.predict(X_test_processed)

# Generate the confusion matrix
matrix = confusion_matrix(y_test, np.argmax(predictions, axis=1))

# Plot the confusion matrix
plt.figure(figsize=(12, 7))
sns.heatmap(matrix, annot=True)
plt.show()
```

```
395/395 [==============================] - 1s 2ms/step
```