Interpret Sign Language with Deep Learning

- Which sign represents which letter in the alphabet?

American Sign Language (ASL) is a natural language that serves as the predominant sign language of Deaf communities in the United States and most of Anglophone Canada. ASL possesses a set of 26 signs known as the American manual alphabet, which can be used to spell out words from the English language.

Import Libraries

In [1]:
```
import keras
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, Lambda, MaxPool2D, BatchNormalization
from keras.utils import np_utils
from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras import models, layers, optimizers
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.utils import class_weight
from keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta, RMSprop
from keras.models import Sequential, model_from_json
from keras.layers import AveragePooling2D
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras import backend as K
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.applications.inception_v3 import InceptionV3
import os
from glob import glob
import matplotlib.pyplot as plt
import random
import cv2
import pandas as pd
import numpy as np
import matplotlib.gridspec as gridspec
import seaborn as sns
import zlib
import itertools
import sklearn
import itertools
import scipy
import skimage
from skimage.transform import resize
import csv
from tqdm import tqdm
from sklearn import model_selection
from sklearn.model_selection import train_test_split, learning_curve,KFold,cross_val_score,StratifiedKFold
from sklearn.utils import class_weight
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

from keras.applications.mobilenet import MobileNet
from sklearn.metrics import roc_auc_score
```

```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

```
/opt/conda/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Load Data

In [2]:
```
# print(os.listdir("../input"))
# print(os.listdir("../input/asl-alphabet"))
# print(os.listdir("../input/asl-alphabet/asl_alphabet_train"))
# print(os.listdir("../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train"))
# print(os.listdir("../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/A"))
```

In [3]:
```
imageSize=50
train_dir = "../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/"
test_dir =  "../input/asl-alphabet/asl_alphabet_test/asl_alphabet_test/"
from tqdm import tqdm
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []
    for folderName in os.listdir(folder):
        if not folderName.startswith('.'):
            if folderName in ['A']:
                label = 0
            elif folderName in ['B']:
                label = 1
            elif folderName in ['C']:
                label = 2
            elif folderName in ['D']:
                label = 3
            elif folderName in ['E']:
                label = 4
            elif folderName in ['F']:
                label = 5
            elif folderName in ['G']:
                label = 6
            elif folderName in ['H']:
                label = 7
            elif folderName in ['I']:
                label = 8
            elif folderName in ['J']:
                label = 9
            elif folderName in ['K']:
                label = 10
            elif folderName in ['L']:
                label = 11
            elif folderName in ['M']:
                label = 12
            elif folderName in ['N']:
```

```
                label = 13
            elif folderName in ['O']:
                label = 14
            elif folderName in ['P']:
                label = 15
            elif folderName in ['Q']:
                label = 16
            elif folderName in ['R']:
                label = 17
            elif folderName in ['S']:
                label = 18
            elif folderName in ['T']:
                label = 19
            elif folderName in ['U']:
                label = 20
            elif folderName in ['V']:
                label = 21
            elif folderName in ['W']:
                label = 22
            elif folderName in ['X']:
                label = 23
            elif folderName in ['Y']:
                label = 24
            elif folderName in ['Z']:
                label = 25
            elif folderName in ['del']:
                label = 26
            elif folderName in ['nothing']:
                label = 27
            elif folderName in ['space']:
                label = 28
            else:
                label = 29
            for image_filename in tqdm(os.listdir(folder + folderName)):
                img_file = cv2.imread(folder + folderName + '/' + image_filename)
                if img_file is not None:
                    img_file = skimage.transform.resize(img_file, (imageSize, imageSize, 3))
                    img_arr = np.asarray(img_file)
                    X.append(img_arr)
                    y.append(label)
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
X_train, y_train = get_data(train_dir)
#X_test, y_test= get_data(test_dir) # Too few images

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2)

# Encode labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
from keras.utils.np_utils import to_categorical
y_trainHot = to_categorical(y_train, num_classes = 30)
y_testHot = to_categorical(y_test, num_classes = 30)
```

```
  0%|          | 0/3000 [00:00<?, ?it/s]/opt/conda/lib/python3.6/site-packages/skimage/transform/_warps.py:84: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
100%|██████████| 3000/3000 [00:25<00:00, 118.63it/s]
100%|██████████| 3000/3000 [00:26<00:00, 115.21it/s]
100%|██████████| 3000/3000 [00:33<00:00, 89.11it/s]
100%|██████████| 3000/3000 [00:25<00:00, 117.54it/s]
100%|██████████| 3000/3000 [00:25<00:00, 116.45it/s]
100%|██████████| 3000/3000 [00:33<00:00, 90.77it/s]
100%|██████████| 3000/3000 [00:25<00:00, 115.87it/s]
100%|██████████| 3000/3000 [00:25<00:00, 116.04it/s]
100%|██████████| 3000/3000 [00:29<00:00, 101.42it/s]
100%|██████████| 3000/3000 [00:26<00:00, 113.64it/s]
100%|██████████| 3000/3000 [00:26<00:00, 112.68it/s]
100%|██████████| 3000/3000 [00:32<00:00, 93.51it/s]
100%|██████████| 3000/3000 [00:26<00:00, 113.72it/s]
100%|██████████| 3000/3000 [00:26<00:00, 114.93it/s]
100%|██████████| 3000/3000 [00:34<00:00, 86.60it/s]
100%|██████████| 3000/3000 [00:25<00:00, 115.57it/s]
100%|██████████| 3000/3000 [00:25<00:00, 115.55it/s]
100%|██████████| 3000/3000 [00:26<00:00, 115.14it/s]
100%|██████████| 3000/3000 [00:26<00:00, 114.03it/s]
100%|██████████| 3000/3000 [00:25<00:00, 116.17it/s]
100%|██████████| 3000/3000 [00:33<00:00, 89.99it/s]
100%|██████████| 3000/3000 [00:33<00:00, 90.21it/s]
100%|██████████| 3000/3000 [00:26<00:00, 114.99it/s]
100%|██████████| 3000/3000 [00:25<00:00, 117.33it/s]
100%|██████████| 3000/3000 [00:25<00:00, 116.82it/s]
100%|██████████| 3000/3000 [00:32<00:00, 91.35it/s]
100%|██████████| 3000/3000 [00:26<00:00, 115.07it/s]
100%|██████████| 3000/3000 [00:33<00:00, 89.63it/s]
100%|██████████| 3000/3000 [00:25<00:00, 115.54it/s]
```

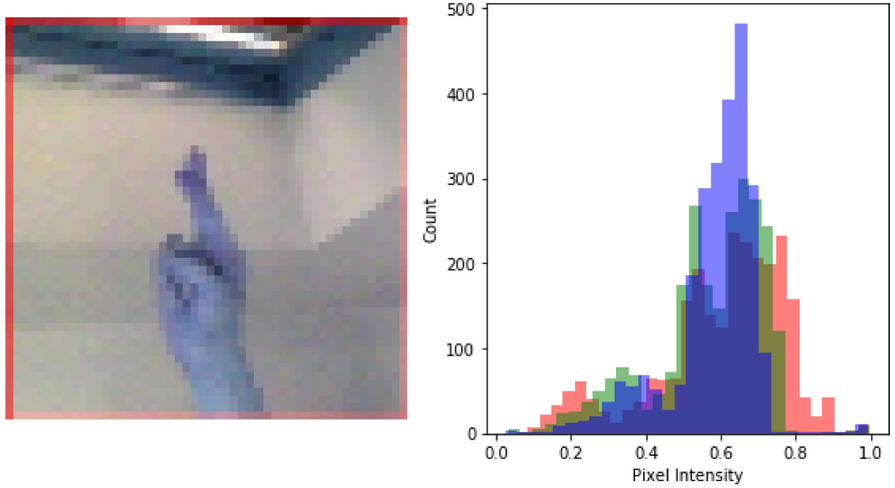
In [4]:
```python
# Shuffle data to permit further subsampling
from sklearn.utils import shuffle
X_train, y_trainHot = shuffle(X_train, y_trainHot, random_state=13)
X_test, y_testHot = shuffle(X_test, y_testHot, random_state=13)
X_train = X_train[:30000]
X_test = X_test[:30000]
y_trainHot = y_trainHot[:30000]
y_testHot = y_testHot[:30000]
```

Vizualize Data

The min/max pixel values are already scaled between 0 and 1

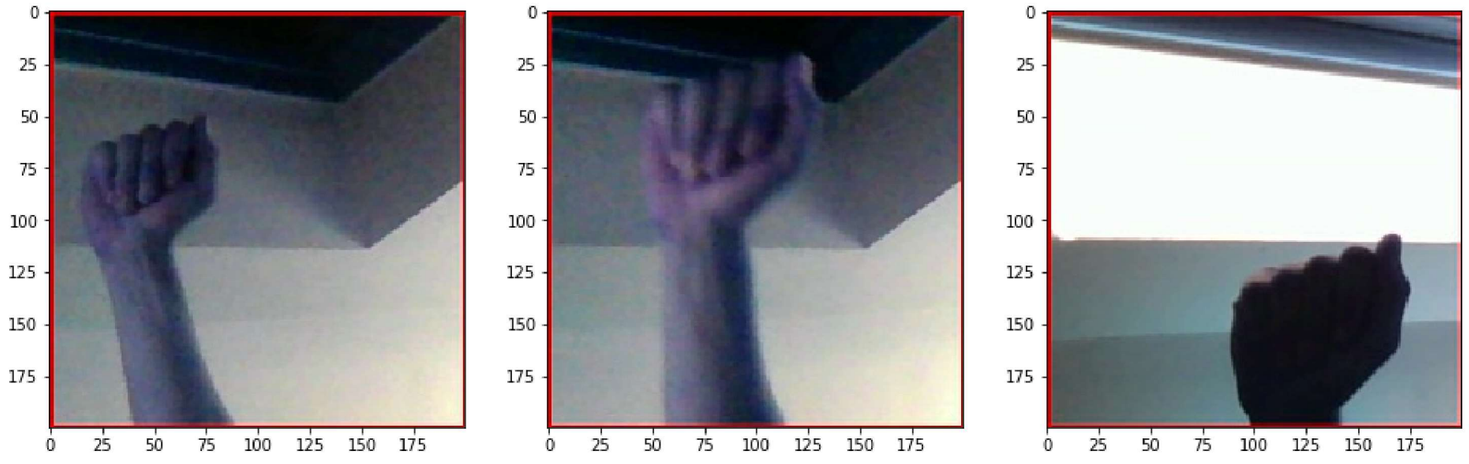
In [5]:
```python
def plotHistogram(a):
    """
    Plot histogram of RGB Pixel Intensities
    """
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.imshow(a)
    plt.axis('off')
    histo = plt.subplot(1,2,2)
    histo.set_ylabel('Count')
    histo.set_xlabel('Pixel Intensity')
    n_bins = 30
    plt.hist(a[:,:,0].flatten(), bins= n_bins, lw = 0, color='r', alpha=0.5);
```

```
    plt.hist(a[:,:,1].flatten(), bins= n_bins, lw = 0, color='g', alpha=0.5);
    plt.hist(a[:,:,2].flatten(), bins= n_bins, lw = 0, color='b', alpha=0.5);
plotHistogram(X_train[1])
```



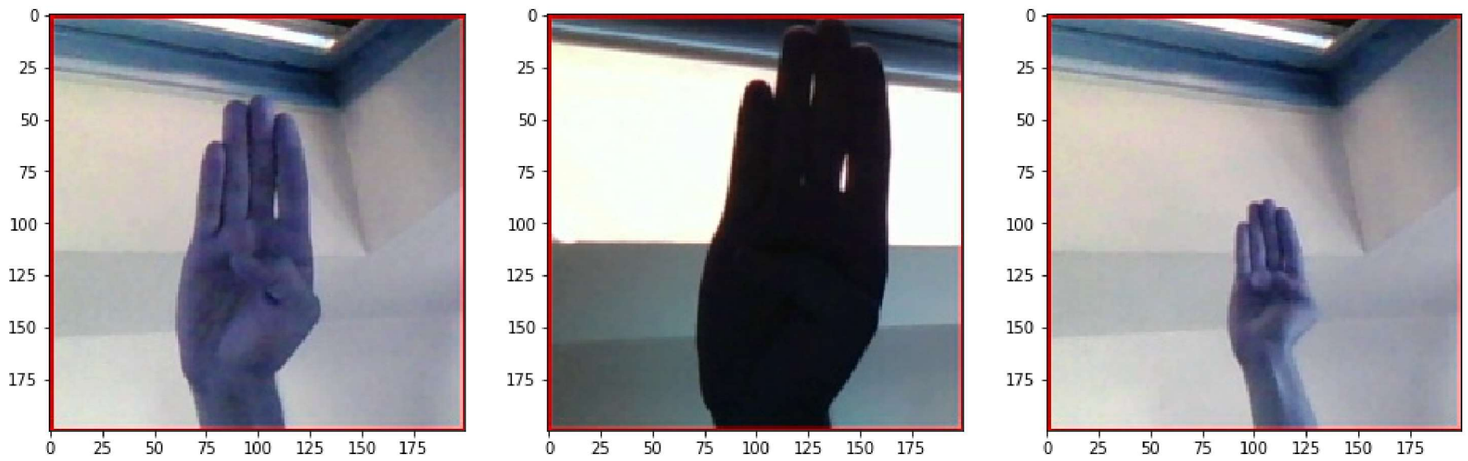
3 images from category "A"

In [6]:
```
multipleImages = glob('../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/A/**')
def plotThreeImages(images):
    r = random.sample(images, 3)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]))
    #;
plotThreeImages(multipleImages)
```



3 images from category "B"

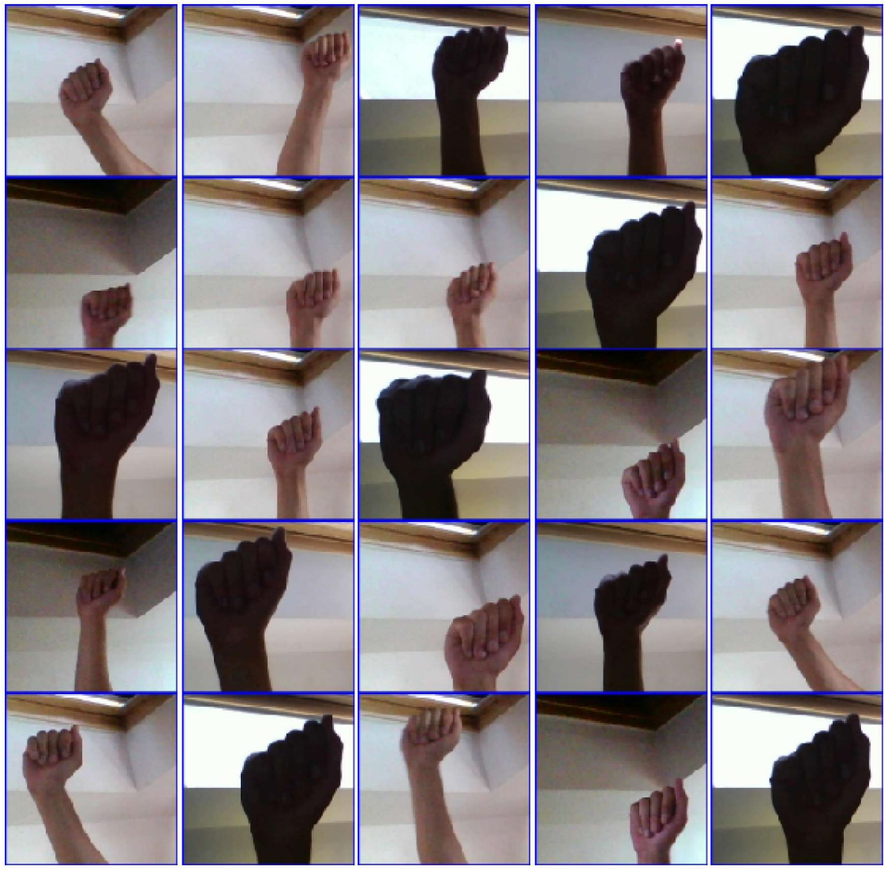
In [7]:
```
multipleImages = glob('../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/B/**')
def plotThreeImages(images):
    r = random.sample(images, 3)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]))
plotThreeImages(multipleImages)
```



20 images from category "A"

In [8]:
```
print("A")
multipleImages = glob('../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/A/**')
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (128, 128))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

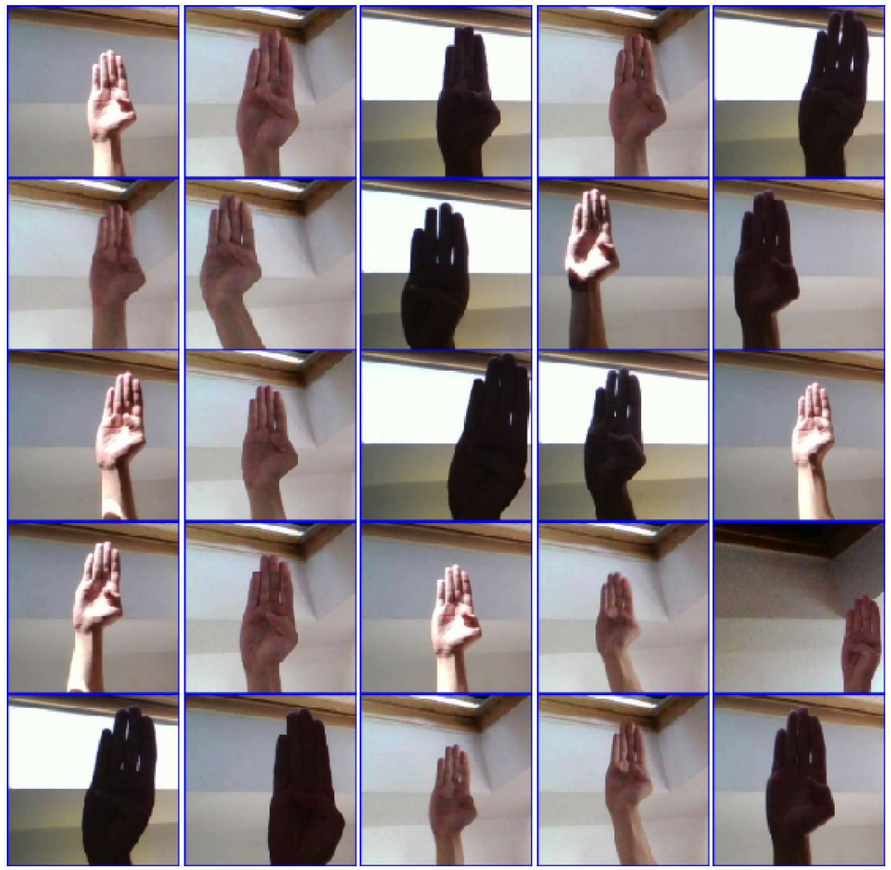
A



20 images from category "B"

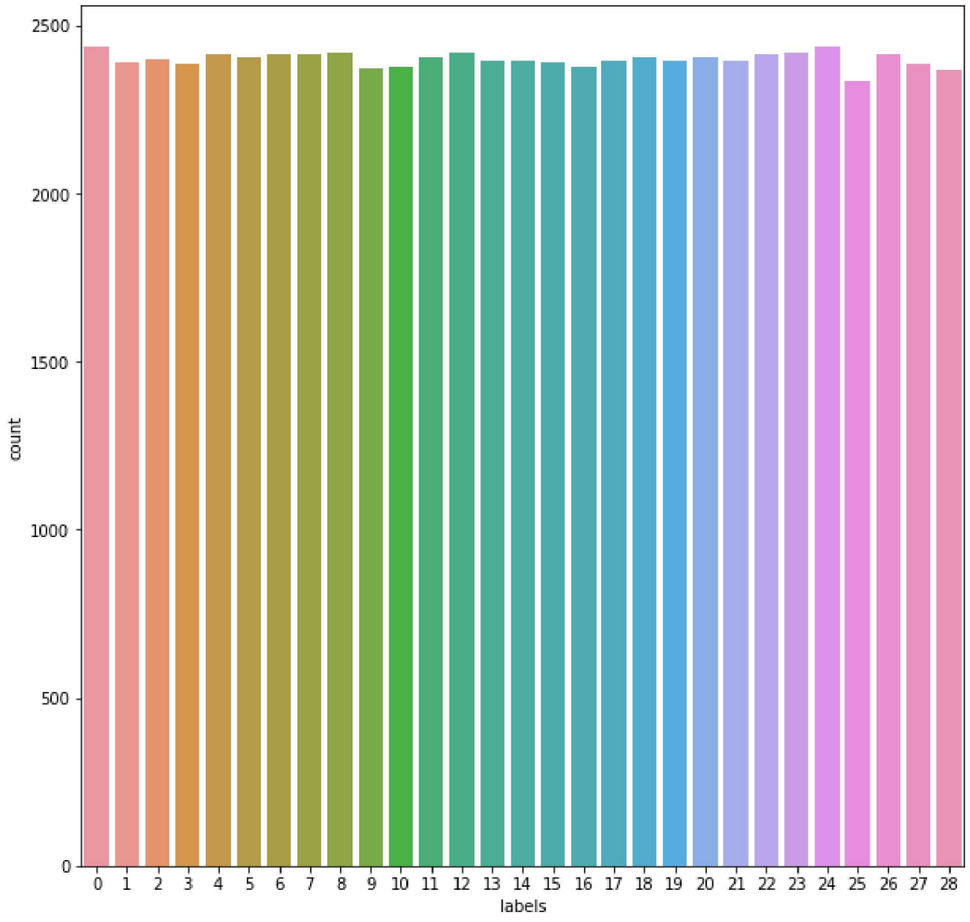
In [9]:
```
print("B")
multipleImages = glob('../input/asl-alphabet/asl_alphabet_train/asl_alphabet_train/B/**')
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (128, 128))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

```
B
```



In [10]:
```
map_characters = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21:
dict_characters=map_characters
import seaborn as sns
df = pd.DataFrame()
df["labels"]=y_train
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)
```

```
{0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 2
3: 'X', 24: 'Y', 25: 'Z', 26: 'del', 27: 'nothing', 28: 'space', 29: 'other'}
```



Define Helper Functions

In [11]:
```
# Helper Functions  Learning Curves and Confusion Matrix

from keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

class MetricsCheckpoint(Callback):
    """Callback that saves metrics after each epoch"""
    def __init__(self, savepath):
        super(MetricsCheckpoint, self).__init__()
        self.savepath = savepath
        self.history = {}
    def on_epoch_end(self, epoch, logs=None):
        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        np.save(self.savepath, self.history)

def plotKerasLearningCurve():
    plt.figure(figsize=(10,5))
    metrics = np.load('logs.npy')[()]
    filt = ['acc'] # try to add 'loss' to see the loss learning curve
```

```
    for k in filter(lambda x : np.any([kk in x for kk in filt]), metrics.keys()):
        l = np.array(metrics[k])
        plt.plot(l, c= 'r' if 'val' not in k else 'b', label='val' if 'val' in k else 'train')
        x = np.argmin(l) if 'loss' in k else np.argmax(l)
        y = l[x]
        plt.scatter(x,y, lw=0, alpha=0.25, s=100, c='r' if 'val' not in k else 'b')
        plt.text(x, y, '{} = {:.4f}'.format(x,y), size='15', color= 'r' if 'val' not in k else 'b')
    plt.legend(loc=4)
    plt.axis([0, None, None, None]);
    plt.grid()
    plt.xlabel('Number of epochs')
    plt.ylabel('Accuracy')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (8,8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

def plot_learning_curve(history):
    plt.figure(figsize=(8,8))
    plt.subplot(1,2,1)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./accuracy_curve.png')
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('./loss_curve.png')
```

Evaluate Classification Models

Transfer learning w/ VGG16 Convolutional Network

In [12]:
```
map_characters1 = map_characters
class_weight1 = class_weight.compute_class_weight('balanced', np.unique(y_train), y_train)
weight_path1 = '../input/keras-pretrained-models/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
weight_path2 = '../input/keras-pretrained-models/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
pretrained_model_1 = VGG16(weights = weight_path1, include_top=False, input_shape=(imageSize, imageSize, 3))
#pretrained_model_2 = InceptionV3(weights = weight_path2, include_top=False, input_shape=(imageSize, imageSize, 3))
optimizer1 = keras.optimizers.Adam()
optimizer2 = keras.optimizers.RMSprop(lr=0.0001)
def pretrainedNetwork(xtrain,ytrain,xtest,ytest,pretrainedmodel,pretrainedweights,classweight,numclasses,numepochs,optimizer,labels):
    base_model = pretrained_model_1 # Topless
    # Add top layer
    x = base_model.output
    x = Flatten()(x)
    predictions = Dense(numclasses, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    # Train top layer
    for layer in base_model.layers:
        layer.trainable = False
    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    callbacks_list = [keras.callbacks.EarlyStopping(monitor='val_acc', patience=3, verbose=1)]
    model.summary()
    # Fit model
    history = model.fit(xtrain,ytrain, epochs=numepochs, class_weight=classweight, validation_data=(xtest,ytest), verbose=1,callbacks = [MetricsCheckpoint('logs')])
    # Evaluate model
    score = model.evaluate(xtest,ytest, verbose=0)
    print('\nKeras CNN - accuracy:', score[1], '\n')
    y_pred = model.predict(xtest)
    print('\n', sklearn.metrics.classification_report(np.where(ytest > 0)[1], np.argmax(y_pred, axis=1), target_names=list(labels.values())), sep='')
    Y_pred_classes = np.argmax(y_pred,axis = 1)
    Y_true = np.argmax(ytest,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plotKerasLearningCurve()
    plt.show()
    plot_learning_curve(history)
    plt.show()
    plot_confusion_matrix(confusion_mtx, classes = list(labels.values()))
    plt.show()
    return model
pretrainedNetwork(X_train, y_trainHot, X_test, y_testHot,pretrained_model_1,weight_path1,class_weight1,30,10,optimizer1,map_characters1)
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 50, 50, 3)	0
block1_conv1 (Conv2D)	(None, 50, 50, 64)	1792
block1_conv2 (Conv2D)	(None, 50, 50, 64)	36928
block1_pool (MaxPooling2D)	(None, 25, 25, 64)	0
block2_conv1 (Conv2D)	(None, 25, 25, 128)	73856
block2_conv2 (Conv2D)	(None, 25, 25, 128)	147584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
block4_conv1 (Conv2D)	(None, 6, 6, 512)	1180160
block4_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block4_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
block4_pool (MaxPooling2D)	(None, 3, 3, 512)	0
block5_conv1 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv2 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv3 (Conv2D)	(None, 3, 3, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 30)	15390

Total params: 14,730,078
Trainable params: 15,390
Non-trainable params: 14,714,688

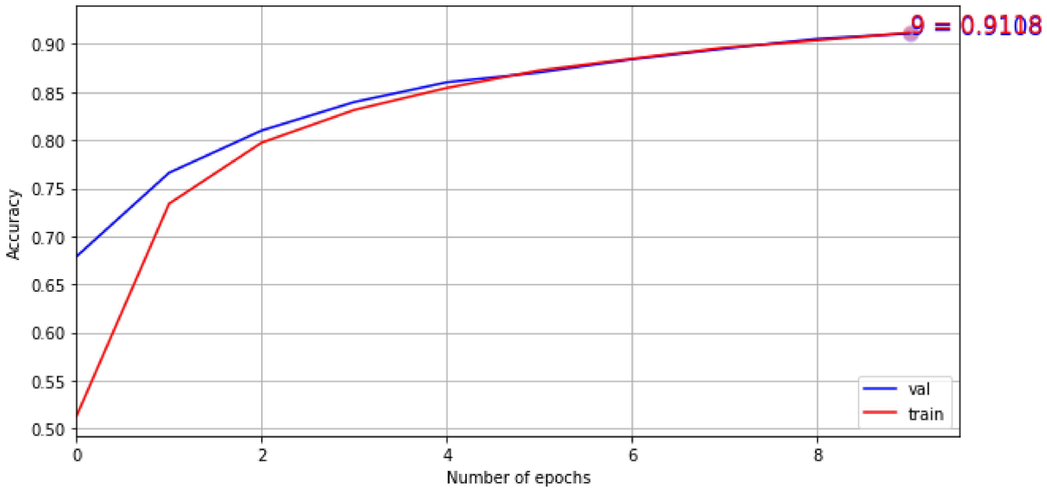
```
Train on 30000 samples, validate on 17400 samples
Epoch 1/10
30000/30000 [==============================] - 20s 654us/step - loss: 2.1969 - acc: 0.5130 - val_loss: 1.5811 - val_acc: 0.6791
Epoch 2/10
30000/30000 [==============================] - 15s 508us/step - loss: 1.3248 - acc: 0.7339 - val_loss: 1.1468 - val_acc: 0.7661
Epoch 3/10
30000/30000 [==============================] - 15s 508us/step - loss: 1.0155 - acc: 0.7974 - val_loss: 0.9250 - val_acc: 0.8101
Epoch 4/10
30000/30000 [==============================] - 15s 509us/step - loss: 0.8391 - acc: 0.8314 - val_loss: 0.7823 - val_acc: 0.8396
Epoch 5/10
30000/30000 [==============================] - 15s 512us/step - loss: 0.7209 - acc: 0.8543 - val_loss: 0.6853 - val_acc: 0.8602
```

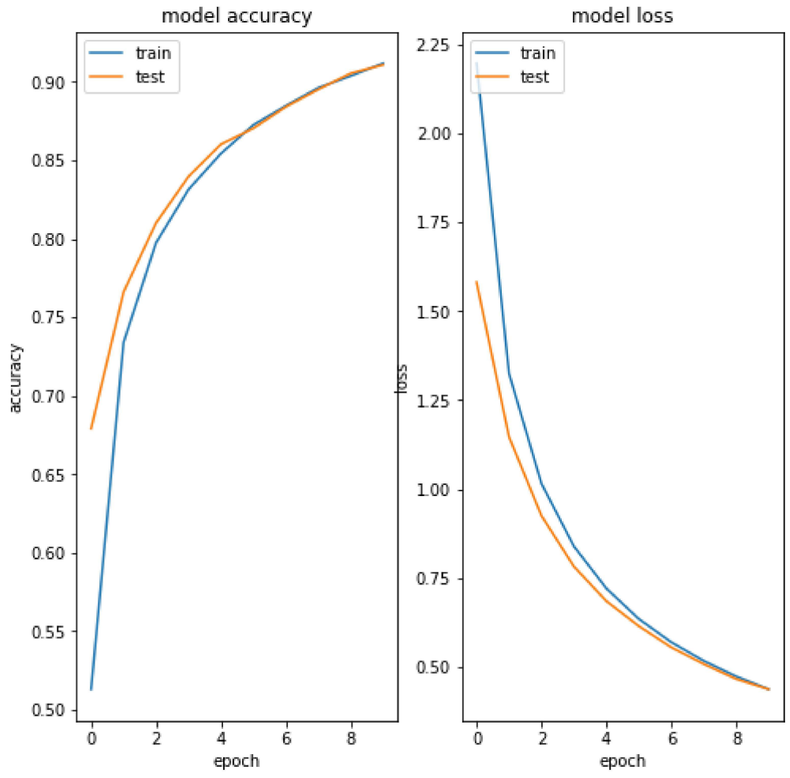
```
Epoch 6/10
30000/30000 [==============================] - 15s 502us/step - loss: 0.6352 - acc: 0.8725 - val_loss: 0.6149 - val_acc: 0.8703
Epoch 7/10
30000/30000 [==============================] - 15s 504us/step - loss: 0.5697 - acc: 0.8847 - val_loss: 0.5552 - val_acc: 0.8840
Epoch 8/10
30000/30000 [==============================] - 15s 506us/step - loss: 0.5174 - acc: 0.8962 - val_loss: 0.5084 - val_acc: 0.8951
Epoch 9/10
30000/30000 [==============================] - 15s 504us/step - loss: 0.4736 - acc: 0.9037 - val_loss: 0.4662 - val_acc: 0.9054
Epoch 10/10
30000/30000 [==============================] - 15s 507us/step - loss: 0.4376 - acc: 0.9118 - val_loss: 0.4375 - val_acc: 0.9108

Keras CNN - accuracy: 0.9108045977011494
```

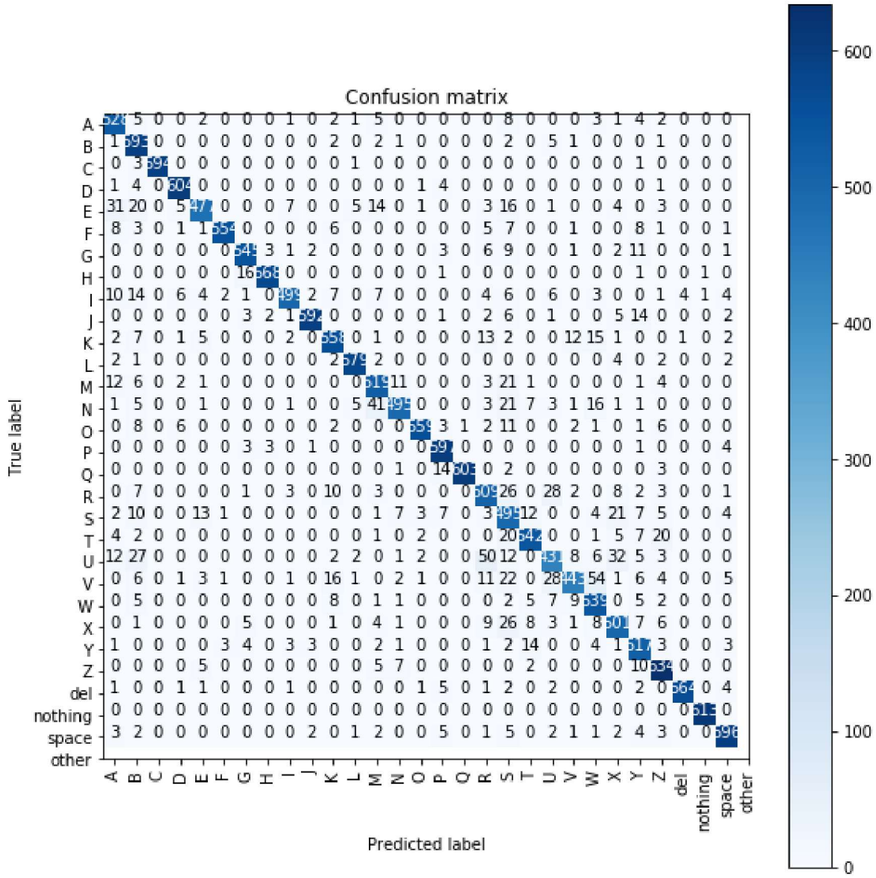
	precision	recall	f1-score	support
A	0.85	0.94	0.89	562
B	0.81	0.98	0.89	608
C	1.00	0.99	1.00	599
D	0.96	0.98	0.97	615
E	0.93	0.81	0.87	587
F	0.99	0.93	0.96	596
G	0.94	0.93	0.94	584
H	0.99	0.97	0.98	587
I	0.96	0.86	0.91	581
J	0.98	0.94	0.96	629
K	0.91	0.90	0.90	622
L	0.97	0.97	0.97	594
M	0.85	0.89	0.87	581
N	0.94	0.82	0.88	602
O	0.98	0.93	0.95	602
P	0.93	0.98	0.96	609
Q	1.00	0.97	0.98	623
R	0.81	0.84	0.83	603
S	0.68	0.83	0.75	595
T	0.92	0.90	0.91	604
U	0.83	0.73	0.78	593
V	0.92	0.73	0.81	606
W	0.82	0.92	0.87	584
X	0.85	0.86	0.86	581
Y	0.84	0.92	0.88	562
Z	0.90	0.96	0.93	663
del	0.99	0.96	0.98	585
nothing	1.00	1.00	1.00	613
space	0.95	0.95	0.95	630
avg / total	0.91	0.91	0.91	17400

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1428: UserWarning: labels size, 29, does not match size of target_names, 30
  .format(len(labels), len(target_names))
```


9 = 0.9108
Accuracy
Number of epochs
val
train


model accuracy
train
test
accuracy
epoch

model loss
train
test
loss
epoch



Out[12]: <keras.engine.training.Model at 0x7f3f9c339f60>

The result shows that we can interpret the signs with an accuracy rate of approximately 91%. That is much better than random chance given that there were 26 different signs.