Reading the Robot Mind

Executive Summary

- Examines one of the most popular public kernels in this contest
- Draws pictures of the internal workings of the neural network
- Explains how these pictures can be used to improve the score
- Proposes this for the Keras feature set

Details

In this contest, as in others, participants struggle to improve their leaderboard (LB) scores by optimizing parameters and methods. One aspect of this optimization is the ability to peer into the internal representation of the trained neural network to let a human expert determine if important information has been lost or is being misinterpreted somewhere along the way. In a recent "AI Explainability Whitepaper" [1] from Google, feature attribution is stressed. This is a wonderful method of determining which features most greatly impact the final classification/coding. Another method uses autoencoders[2] as a data compression means which also can point to flaws in a lossy internal representation of the data.

This Jupyter notebook proposes a third method to read the robot mind. It uses a reverse calculation and attempts to recreate the original data (in this case, the handwritten grapheme) from an internal state of the neural network. In this way, an expert (or any child with 10+ years of schooling in reading and writing Bengali) can clearly see if important information has been lost, what was lost, and even what layer within the neural network it was lost.

Please support by upvoting

If you feel this is an important area for further research, please upvote this notebook. This will help bring attention to seeing these or similar functions get incorporated within the Keras framework, and wherever it seems it will do the most good for future researchers. Perhaps others have come to the same conclusions and are already working similar efforts; in which case I am also happy to help where I can. I am a full time teacher, and as such, I can only devote spare time to this effort. Nevertheless, please feel free to reach out to me in this regard.

references

- [1] - https://cloud.google.com/ml-engine/docs/ai-explanations/overview (https://cloud.google.com/ml-engine/docs/ai-explanations/overview)
- [2] - https://en.wikipedia.org/wiki/Autoencoder (https://en.wikipedia.org/wiki/Autoencoder)

It also builds upon work I have published in US Patents and my 2013 PhD Dissertation

- Method and apparatus for developing a neural network for phoneme recognition US 5,749,066
- Method and apparatus for interfacing and training a neural network for phoneme recognition US 5,809,462
- 2013 Signal Processing of EEG for the Detection of Attentiveness towards Short Training Videos https://scholarscompass.vcu.edu/etd/558/ (https://scholarscompass.vcu.edu/etd/558/)

PANv00 - Forked from the notebook "Bengali Graphemes: Starter EDA+ Multi Output CNN" https://www.kaggle.com/kaushal2896/bengali-graphemes-starter-eda-multi-output-cnn (https://www.kaggle.com/kaushal2896/bengali-graphemes-starter-eda-multi-output-cnn) Courtesy Kaushal Shah [saved trained model for use in this "no GPU" analysis kernel] LEADERBOARD SCORE of - 0.9353

Pointer to version 00 https://www.kaggle.com/pnussbaum/grapheme-mind-reader-panv00 (https://www.kaggle.com/pnussbaum/grapheme-mind-reader-panv00)

PANv01 through v12 - Added "mind reader" code and removed extraneous code. Now the notebook reads in a trained model (from v00) and creates additional neural netowrks that are subsets of the trained NN in order to look into the internal representations of the convolutional layers. Namely... graphical images of the kernels themselves, as well as "reverse processing" of the output of internal layers, back to an estimation of the original image - to look for flaws.

- PANv12d-e - Corrected terminology used in comments. Corrected various bugs
- PANv12f - Simplified code. Now shows all convolutional layers

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from tqdm.auto import tqdm
from glob import glob
import time, gc
import cv2

from tensorflow import keras
import matplotlib.image as mpimg
from keras.preprocessing.image import ImageDataGenerator
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import PIL.Image as Image, PIL.ImageDraw as ImageDraw, PIL.ImageFont as ImageFont
from matplotlib import pyplot as plt
import seaborn as sns
```

In [2]:

```
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the inp
ut directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

In [3]:
```
train_df_ = pd.read_csv('/kaggle/input/bengaliai-cv19/train.csv')
test_df_ = pd.read_csv('/kaggle/input/bengaliai-cv19/test.csv')
class_map_df = pd.read_csv('/kaggle/input/bengaliai-cv19/class_map.csv')
sample_sub_df = pd.read_csv('/kaggle/input/bengaliai-cv19/sample_submission.csv')
```

In [4]:
```
print(f'Size of training data: {train_df_.shape}')
print(f'Size of test data: {test_df_.shape}')
print(f'Size of class map: {class_map_df.shape}')
```

In [5]:
```
# remove the picture of the "correct" grapheme - not relevant to the contest
train_df_ = train_df_.drop(['grapheme'], axis=1, inplace=False)
print(f'Size of training data: {train_df_.shape}')
```

In [6]:
```
# Make the correct classifications, unsigned 8-bit integers
train_df_[['grapheme_root', 'vowel_diacritic', 'consonant_diacritic']] = train_df_[['grapheme_root',
'vowel_diacritic', 'consonant_diacritic']].astype('uint8')
print(f'Size of training data: {train_df_.shape}')
```

In [7]:
```
# the images are resized to 64 pixels square
IMG_SIZE=64
# the images have only black and white (one color channel)
N_CHANNELS=1
print("image size is ", IMG_SIZE, " x ", IMG_SIZE, " with a color depth of ", N_CHANNELS)
```

In [8]:
```
def resize(df, size=IMG_SIZE, need_progress_bar=True):
    resized = {}
    if need_progress_bar:
        for i in tqdm(range(df.shape[0])):
            # read in an image from the storage directory and resize it
            image = cv2.resize(df.loc[df.index[i]].values.reshape(137,236),(size,size))
            resized[df.index[i]] = image.reshape(-1)
    else:
        for i in range(df.shape[0]):
            image = cv2.resize(df.loc[df.index[i]].values.reshape(137,236),(size,size))
            resized[df.index[i]] = image.reshape(-1)
    resized = pd.DataFrame(resized).T
    return resized

print("The function resize() reads images from the storage directory and resizes them")
```

In [9]:
```
def get_dummies(df):
    cols = []
    for col in df:
        cols.append(pd.get_dummies(df[col].astype(str)))
    return pd.concat(cols, axis=1)

print("The function get_dummies() converts categorical variables into dummy/indicator variables.")
```

Basic Model

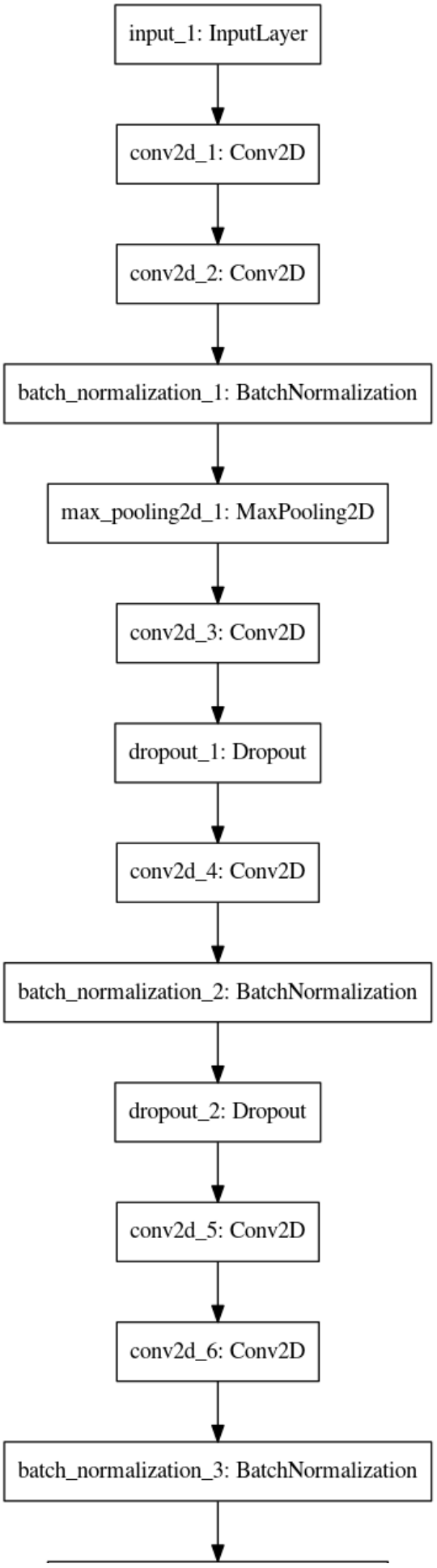
In [10]:
```
# Load the model trained in version 00
# NOTE: You must first select "File" then "add or upload data" from the public kernel available here:
# https://www.kaggle.com/pnussbaum/grapheme-mind-reader-panv00
model = load_model('/kaggle/input/grapheme-mind-reader-panv00/Bengali_Graphemes_K_Shah.h5')
```

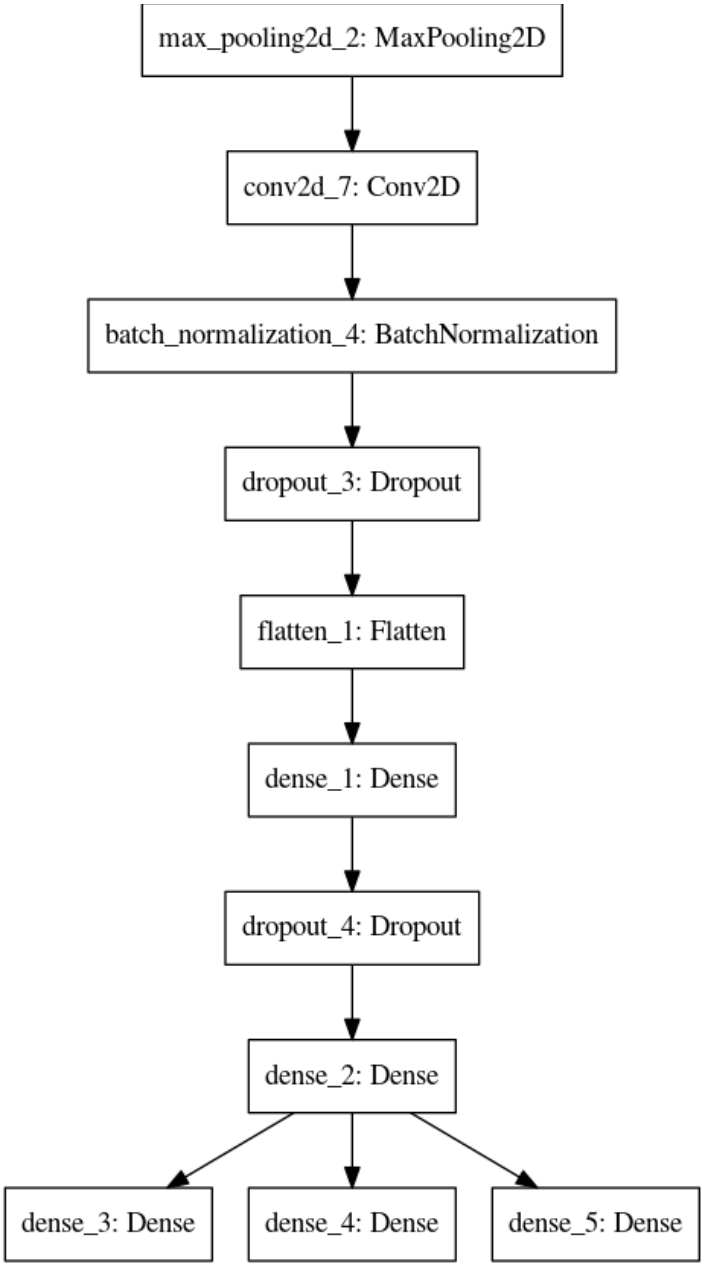
In [11]:
```
model.summary()
```

In [12]:

```
# Visuallize the network
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```

Out[12]:


input_1: InputLayer
conv2d_1: Conv2D
conv2d_2: Conv2D
batch_normalization_1: BatchNormalization
max_pooling2d_1: MaxPooling2D
conv2d_3: Conv2D
dropout_1: Dropout
conv2d_4: Conv2D
batch_normalization_2: BatchNormalization
dropout_2: Dropout
conv2d_5: Conv2D
conv2d_6: Conv2D
batch_normalization_3: BatchNormalization


max_pooling2d_2: MaxPooling2D
conv2d_7: Conv2D
batch_normalization_4: BatchNormalization
dropout_3: Dropout
flatten_1: Flatten
dense_1: Dense
dropout_4: Dropout
dense_2: Dense
dense_3: Dense
dense_4: Dense
dense_5: Dense

In [13]:
```
print("Loading the training set...")
# Just take the first Parquet for speed
train_df = pd.merge(pd.read_parquet(f'/kaggle/input/bengaliai-cv19/train_image_data_{0}.parquet'), tr
ain_df_, on='image_id').drop(['image_id'], axis=1)

X_train = train_df.drop(['grapheme_root', 'vowel_diacritic', 'consonant_diacritic'], axis=1)
X_train = resize(X_train)/255

# CNN takes images in shape `(batch_size, h, w, channels)`, so reshape the images
X_train = X_train.values.reshape(-1, IMG_SIZE, IMG_SIZE, N_CHANNELS)

Y_train_root = pd.get_dummies(train_df['grapheme_root']).values
Y_train_vowel = pd.get_dummies(train_df['vowel_diacritic']).values
Y_train_consonant = pd.get_dummies(train_df['consonant_diacritic']).values

print("...done loading the training set.\n")

print(f'Training images: {X_train.shape}')
print(f'Training labels root: {Y_train_root.shape}')
print(f'Training labels vowel: {Y_train_vowel.shape}')
print(f'Training labels consonants: {Y_train_consonant.shape}')
```

```
100%                                  50210/50210 [01:23<00:00, 599.07it/s]
```

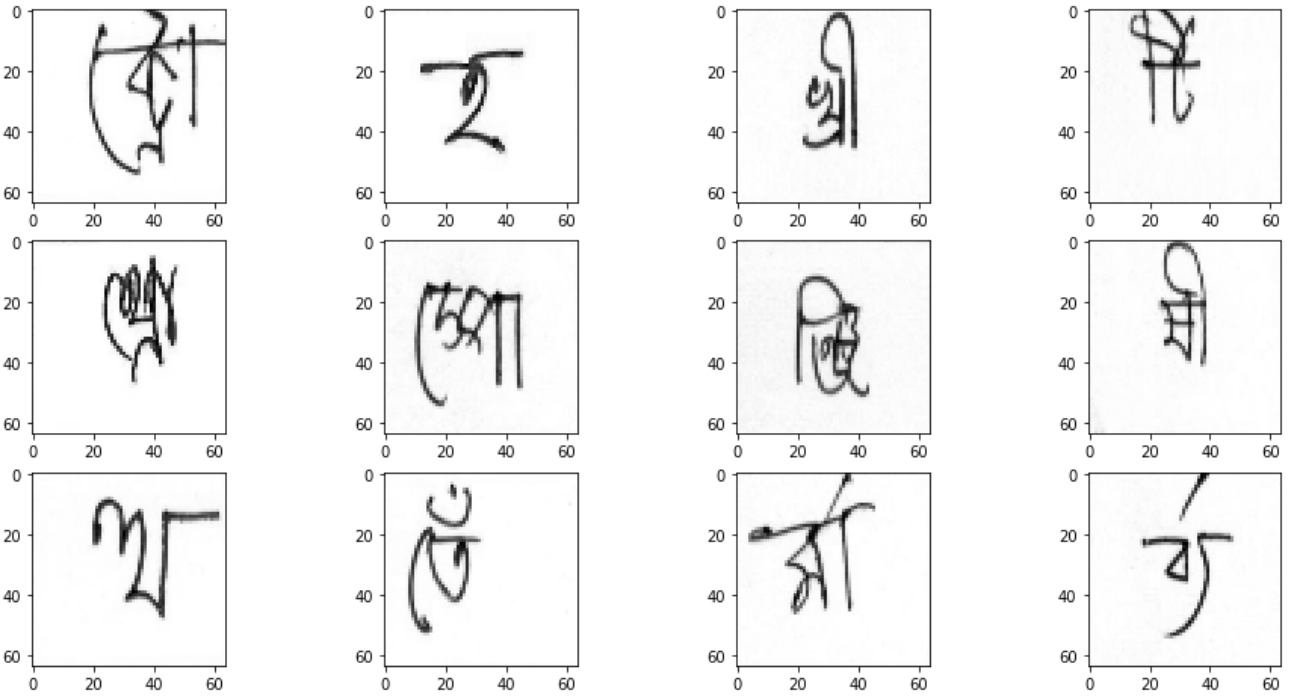
In [14]:
```
print("Let's see how the model behaves on the training set")

one_pred = model.predict(X_train)

print("Have classified ", X_train.shape[0], "training images")
```

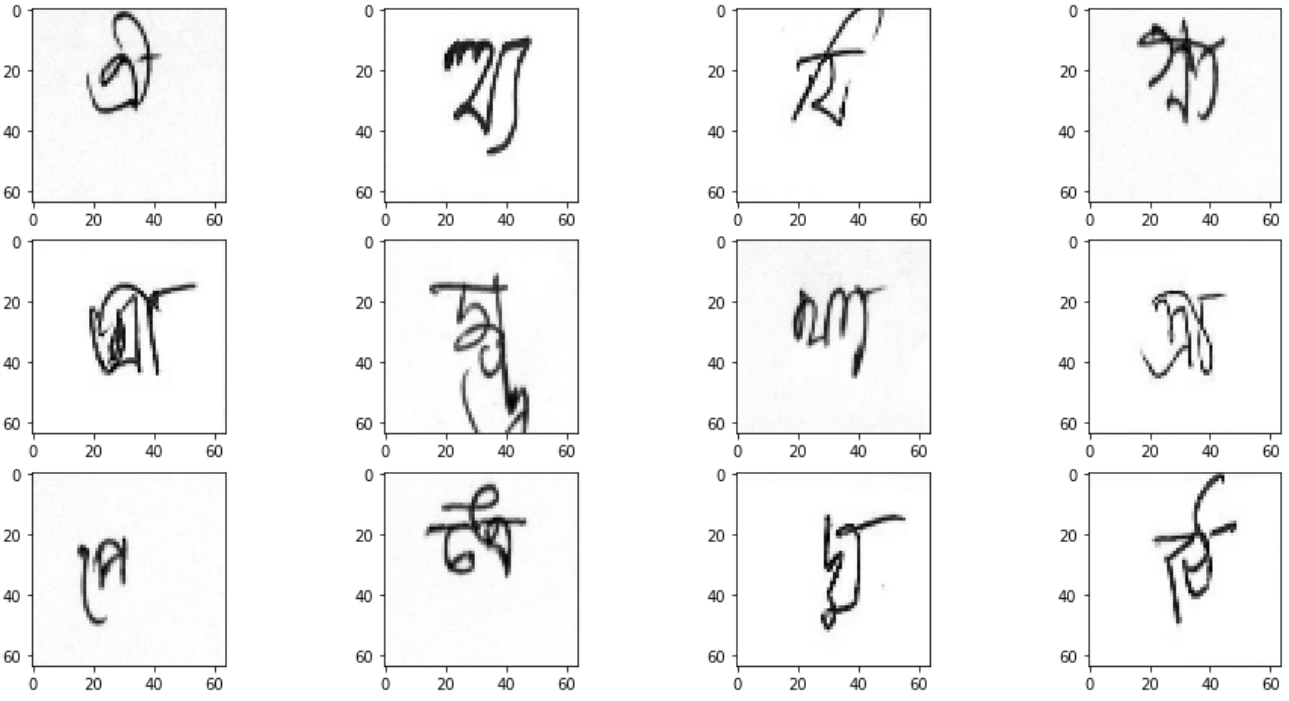
In [15]:

```
print("Training input shape:" , X_train.shape)
print("Root output shape:" , one_pred[0].shape)
print("Vowel output shape:" , one_pred[1].shape)
print("Consonant output shape:" , one_pred[2].shape)
all_correct = []
all_incorrect = []
for i in range(X_train.shape[0]):
    if (np.argmax(Y_train_root[i]) != np.argmax(one_pred[0][i]) and
        np.argmax(Y_train_vowel[i]) != np.argmax(one_pred[1][i]) and
        np.argmax(Y_train_consonant[i]) != np.argmax(one_pred[2][i])) :
        all_incorrect.append(i)
    if (np.argmax(Y_train_root[i]) == np.argmax(one_pred[0][i]) and
        np.argmax(Y_train_vowel[i]) == np.argmax(one_pred[1][i]) and
        np.argmax(Y_train_consonant[i]) == np.argmax(one_pred[2][i])) :
        all_correct.append(i)

# for i, j in enumerate(all_incorrect) :
#     print(i, j)
print("Total images examined ", X_train.shape[0])
print("Total images correctly classified in all three areas ", len(all_correct))
print("Total images incorrectly classified in all three areas ", len(all_incorrect))
```

In [16]:

```
# Visualize few samples of current training dataset
print("Now create a set for mind reading - just some examples from the completely correct and complet
ely incorrect")

print("A dozen examples of completely correct training samples")
fig, ax = plt.subplots(nrows=3, ncols=4, figsize=(16, 8))
count=0
X_correct = np.zeros([12,X_train.shape[1], X_train.shape[2], X_train.shape[3]])
for row in ax:
    for col in row:
        col.imshow(X_train[all_correct[count],:,:,0], cmap="gray")
        X_correct[count] = np.copy(X_train[all_correct[count]])
        count += 1
plt.show()

print("A dozen examples of completely INCORRECT training samples")
fig, ax = plt.subplots(nrows=3, ncols=4, figsize=(16, 8))
count=0
X_incorrect = np.zeros([12,X_train.shape[1], X_train.shape[2], X_train.shape[3]])
for row in ax:
    for col in row:
        col.imshow(X_train[all_incorrect[count],:,:,0], cmap="gray")
        X_incorrect[count] = np.copy(X_train[all_incorrect[count]])
        count += 1
plt.show()
```





In [17]:
```
print("These are the images we will examine more closely", X_correct.shape, X_incorrect.shape)
```

Now let's do some mind reading

Given only the output of one of the convolutional layers - is enough information being kept for subsequent layers to process correctly?

To determine this...

We need to process only the output of each convolutional layer, and then reverse the process to create an image. We can then compare that to the original image to seee if too much information has been discarded. Also a human expert can determine if sufficient information has been retained to perform the classification task.

In [18]:

```
# PANv12f - cleaned up the code to support any number of convolutional layers
# NOTE: This "importing of some weights" section uses apriori knowledge of the model that we are trying
to make subsets of
# FUTURE WORK: Make this importing of NN subsets automatic (without apriori knowledge)
# Let's make a new networks whose outputs are the convolution layers:
# NOTE This is manually configured. Need to make this automated.
# the seven Convolutional layers are layers:
# 1, 2, 5, 7, 10, 11, and 14
num_conv = 7 # PANv12f - added number of convolutional layers
conv_layer = [1, 2, 5, 7, 10, 11, 14] # the layer in which the convolutional filter data is stored
scale_size = [1, 1, 2, 1, 1, 1, 2] # any scaling (due to strides or max pooling) that took place just
 prior to or at that conv layer (not cumulative)
model_explore = [] # list of models that stop at each convolutional layer
Conv2D_weights = [] # list of filters for convolutional layer
Conv2D_biases = [] # list of filter biases for convolutional layer

for i in range (num_conv):
    conv_ind = conv_layer[i]
    print("Conv2D ", i, "occurs at layer",conv_ind)
    model_explore.append(Model(inputs=model.inputs, outputs=model.get_layer(index=conv_ind).output))
    print("Including prior layers, will get an Input Shape of:",model_explore[i].input_shape)
    print("and have an Output Shape of:",model_explore[i].output_shape )
    Conv2D_weights.append(np.copy(model.get_layer(index=conv_ind).get_weights()[0]))
    Conv2D_biases.append(np.copy(model.get_layer(index=conv_ind).get_weights()[1]))
    # These are the filters
    print("Final Conv2D has the following rows, columns, color depth, number of filters", Conv2D_weig
hts[i].shape)
    # These are the biases
    print("and bias for each filter", Conv2D_biases[i].shape)
    print("------------------------")
```

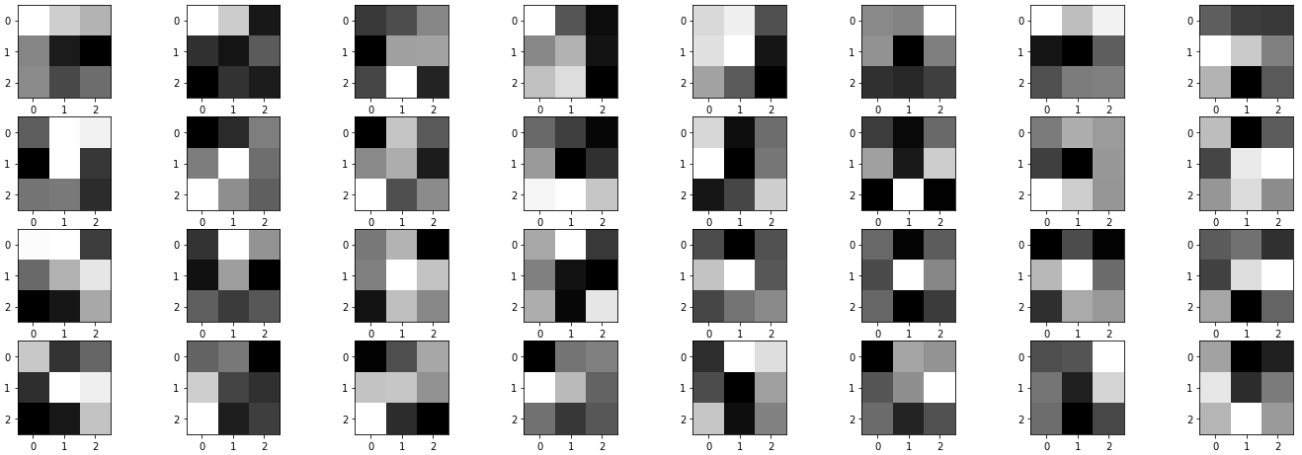
In [19]:

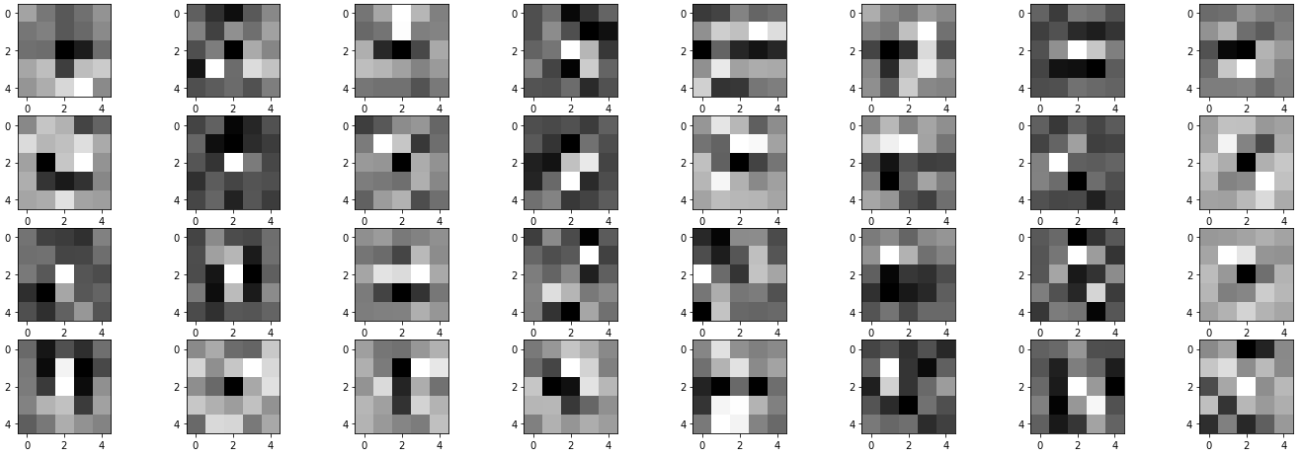
```
# PANv12f now we use a list of models that can be traversed.
for conv_l in range(num_conv):
    print("Overall Explore ",conv_l," Model Summary:", model_explore[conv_l].summary(), "\n")
```

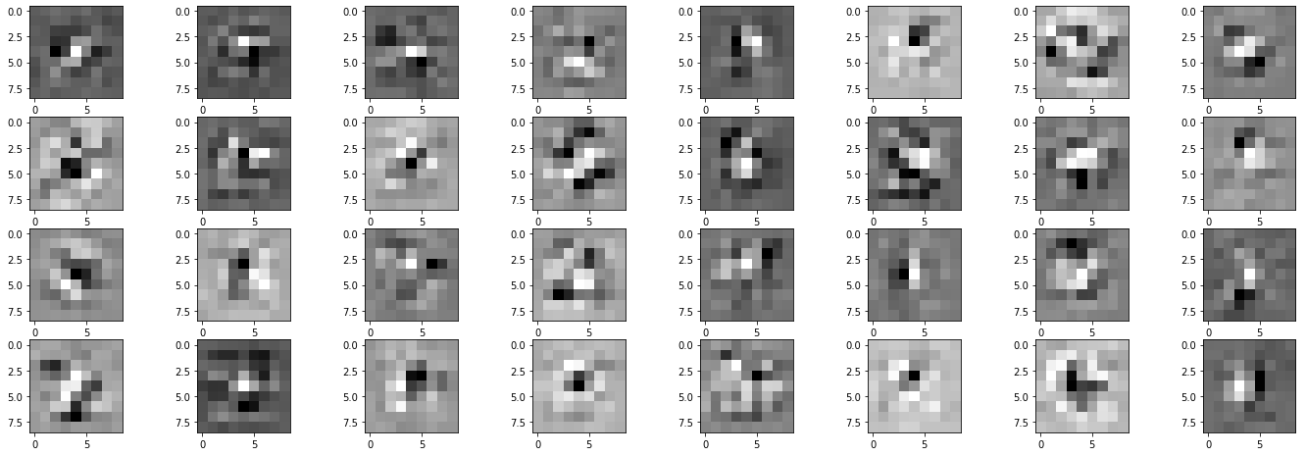
In [20]:

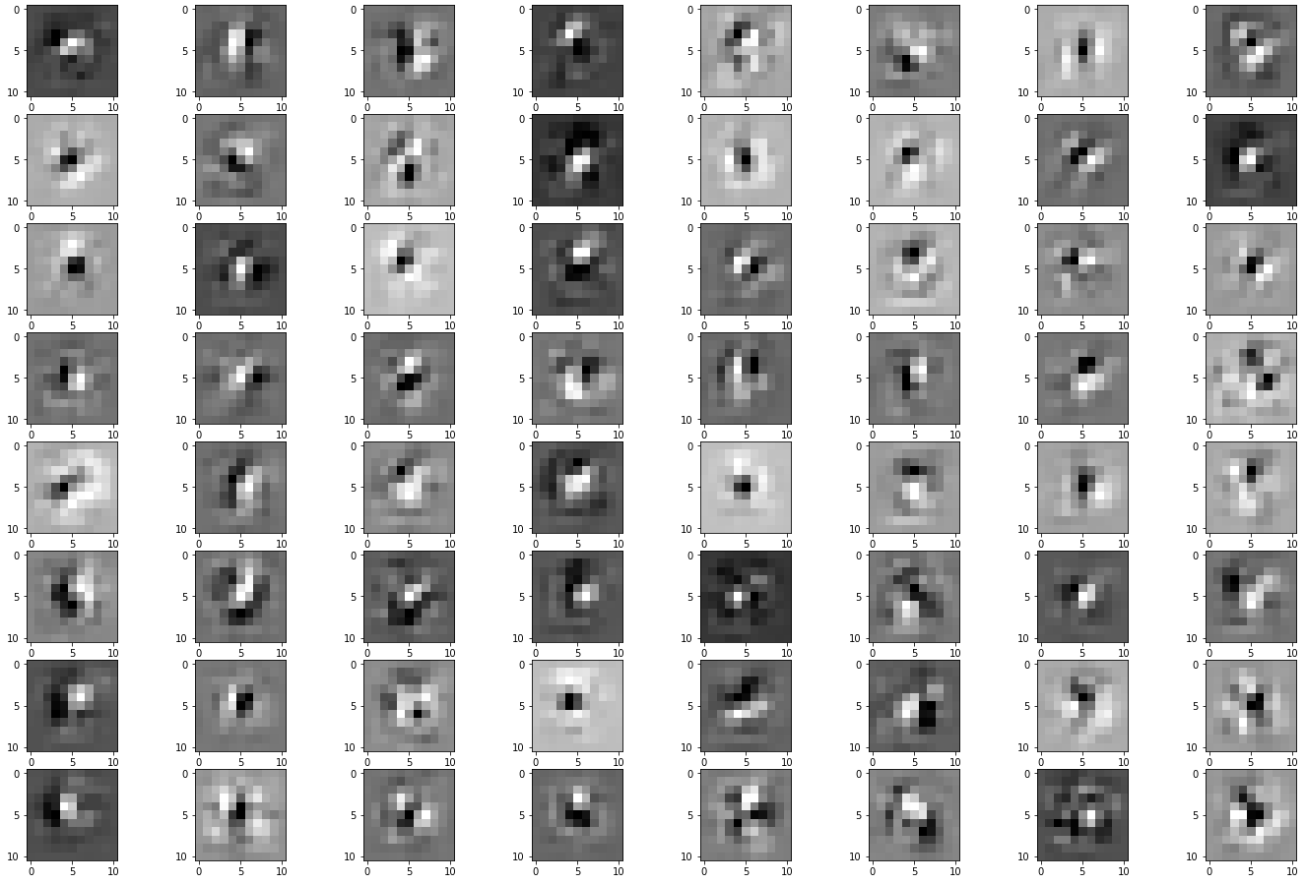
```
# PANv12f - now have a list of filter patches, one for each convolutional layer
num_filt = []            # number of filters in each Conv2D layer
filt_siz = []            # the size of each filter (assume they are square for now)
border_siz = []          # the size of the border around the center pixel of the square (assume filters
are odd numbered dimensions like 3, 5, 7, etc.)
cumulative_border = []   # The cumulative effect of the borders from prior Conv3D layers
filter_patches = []      # The "mind reading" visualization of each filter - equal to the filter at the
first Conv2D layer, but more complex later

for i in range (num_conv):
    conv_ind = conv_layer[i]
    print("Show the reconstructed image patch for each filter of convolutional layer ", i, "which occ
ured at layer ",conv_ind, "in the original model")
    num_filt.append(Conv2D_weights[i].shape[3])
    # filter Size
    filt_siz.append(Conv2D_weights[i].shape[0])
    # Border size
    border_siz.append(int(filt_siz[i] / 2))
    cb = 0
    if i == 0 :
        cumulative_border.append(0)
    else:
        for j in range(i) :
            cb += border_siz[j]
        cumulative_border.append(cb)
    print("Conv Layer", i, " filter Size:", filt_siz[i], " x ", filt_siz[i],
          " and border size (padding) of ", border_siz[i], " and cumulative border from prior layers
 of ", cumulative_border[i])
    filter_patches.append(np.zeros((filt_siz[i] + 2 * cumulative_border[i],
                                    filt_siz[i] + 2 * cumulative_border[i],
                                    num_filt[i])))

    # For now, draw eight filters on a row for display purposes
    this_num_filt = num_filt[i]
    if (this_num_filt % 8 == 0) :
        drows = int(this_num_filt/8)
    else :
        drows = int(this_num_filt/8) + 1
    dcols = 8
    width = 24
    height = int(this_num_filt / 4)
    fig, ax = plt.subplots(nrows=drows, ncols=dcols, figsize=(width, height))

    thisrow = 0
    thiscol = 0
    # temporary variables for filter size and cumulative border size
    cb = cumulative_border[i]
    siz = filt_siz[i]
    image_patch = np.zeros((siz + 2 * cb, siz + 2 * cb))
    for this_filter in range(this_num_filt) :
        if i == 0 :
            image_patch = np.copy(Conv2D_weights[i][:, :, 0, this_filter])
        else:
            image_patch.fill(0)
            for x in range(cb, siz + cb, 1) :
```

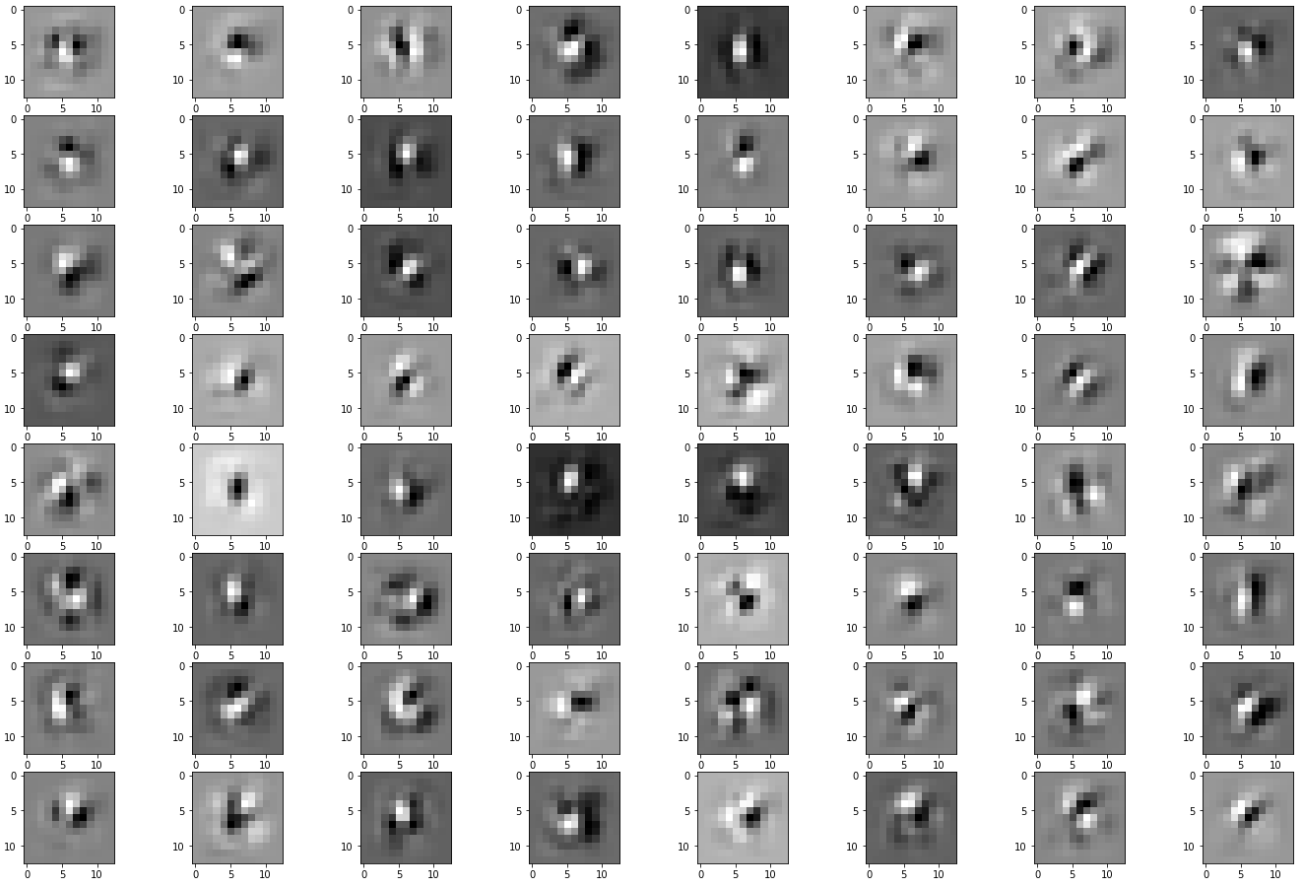
```
                for y in range(cb, siz + cb, 1) :
                    for depth in range(num_filt[i-1]) :
                        image_patch[x-cb:x+cb+1, y-cb:y+cb+1] += np.copy((Conv2D_weights[i][x-cb, y-cb, depth, this_filter] *
                                                                          filter_patches[i-1][:,:,depth]))

        filter_patches[i][:, :, this_filter] = np.copy(image_patch)

        # display the filter
        if (drows > 1) :
            ax[thisrow, thiscol].imshow(filter_patches[i][:, :, this_filter], cmap="gray")
        else :
            ax[thiscol].imshow(filter_patches[i][:, :, this_filter], cmap="gray")
        thiscol += 1
        if (thiscol >=8) :
            thisrow += 1
            thiscol = 0

    plt.show()
```

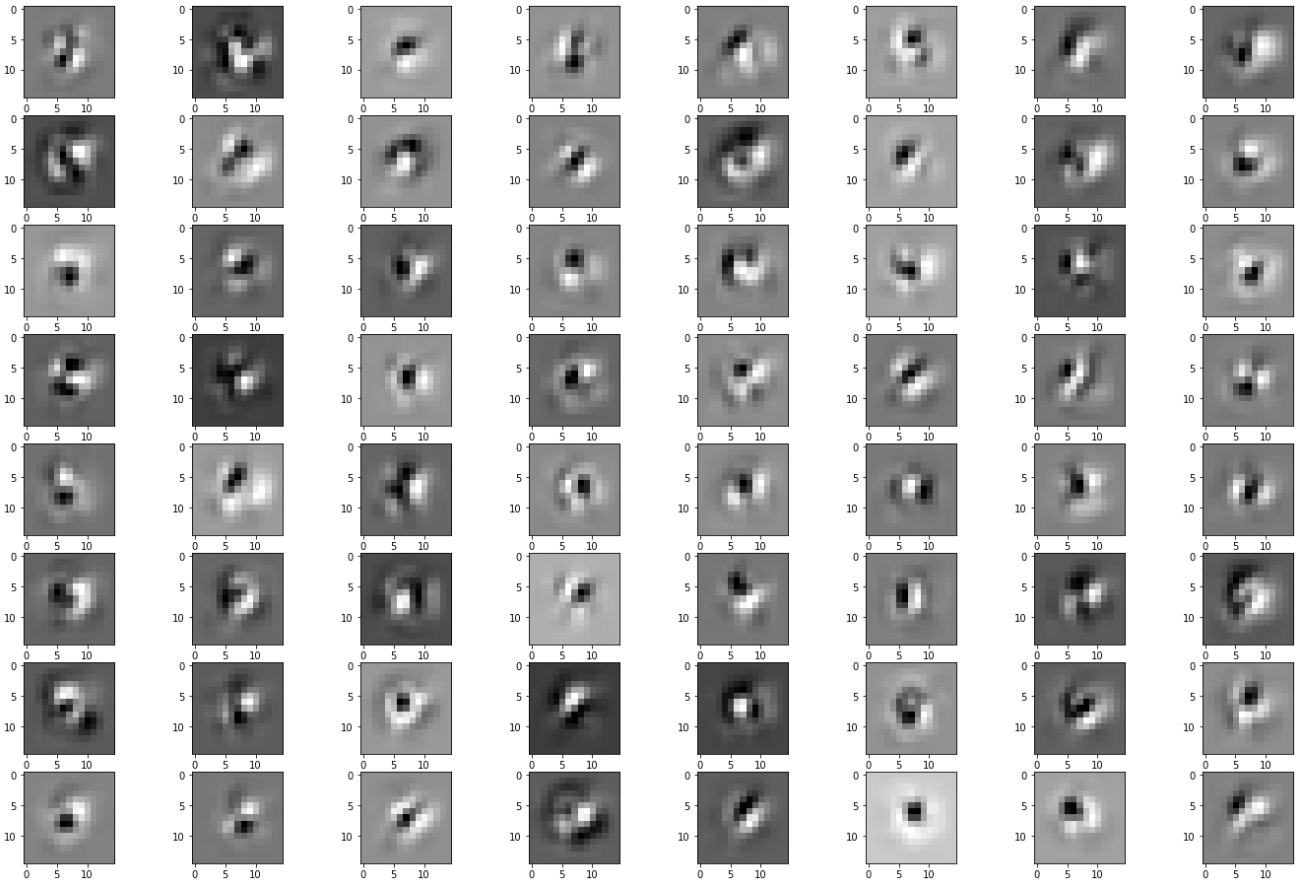


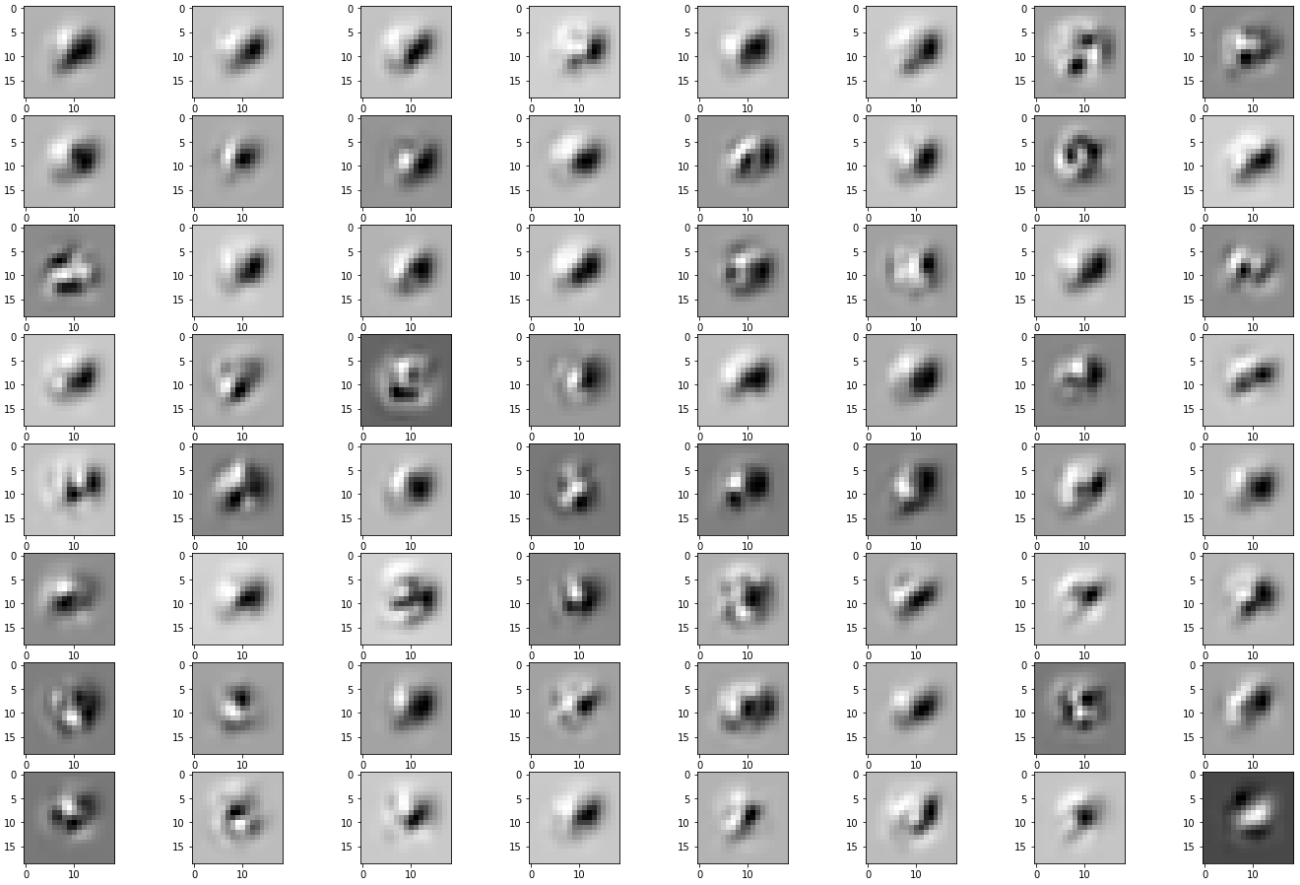










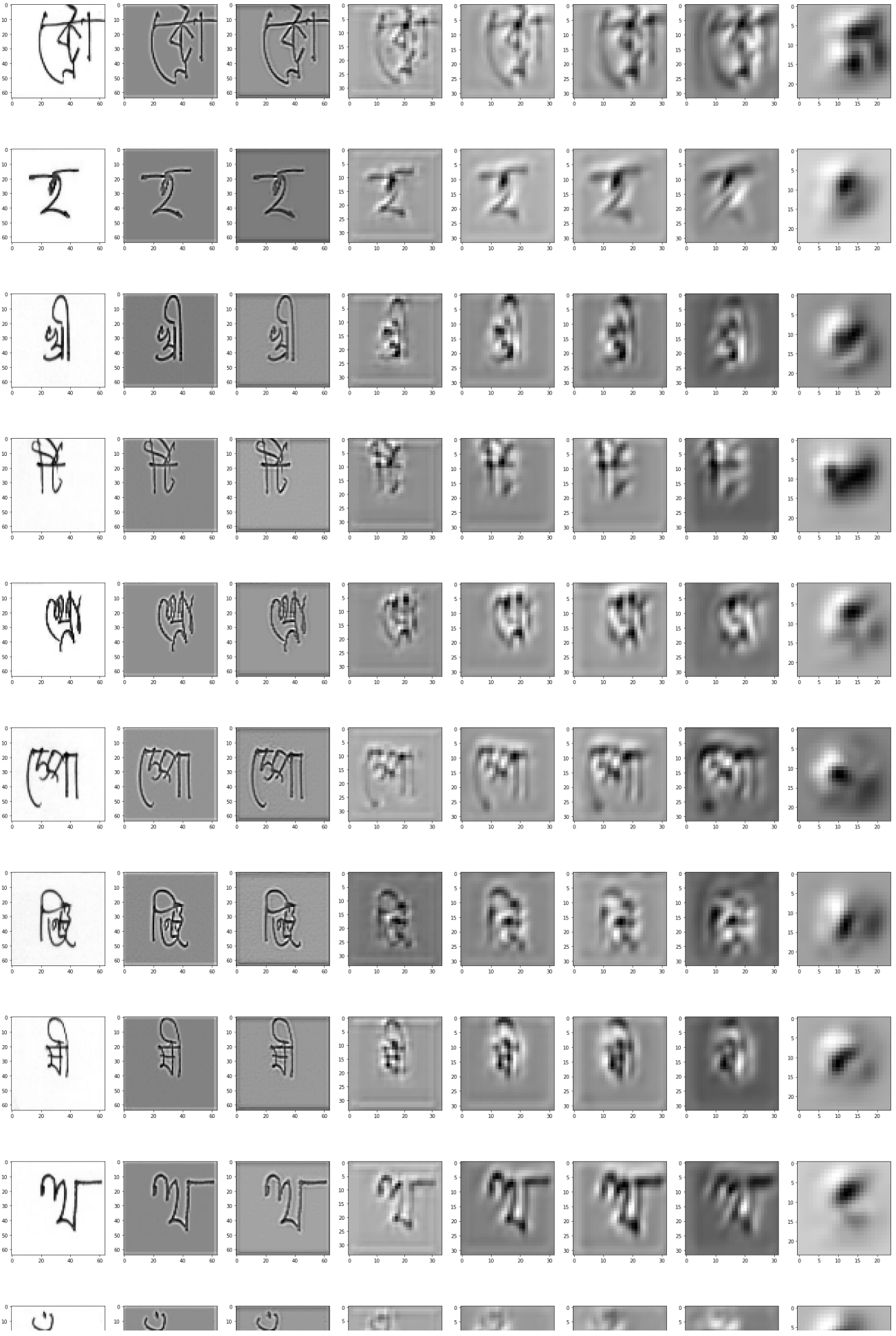


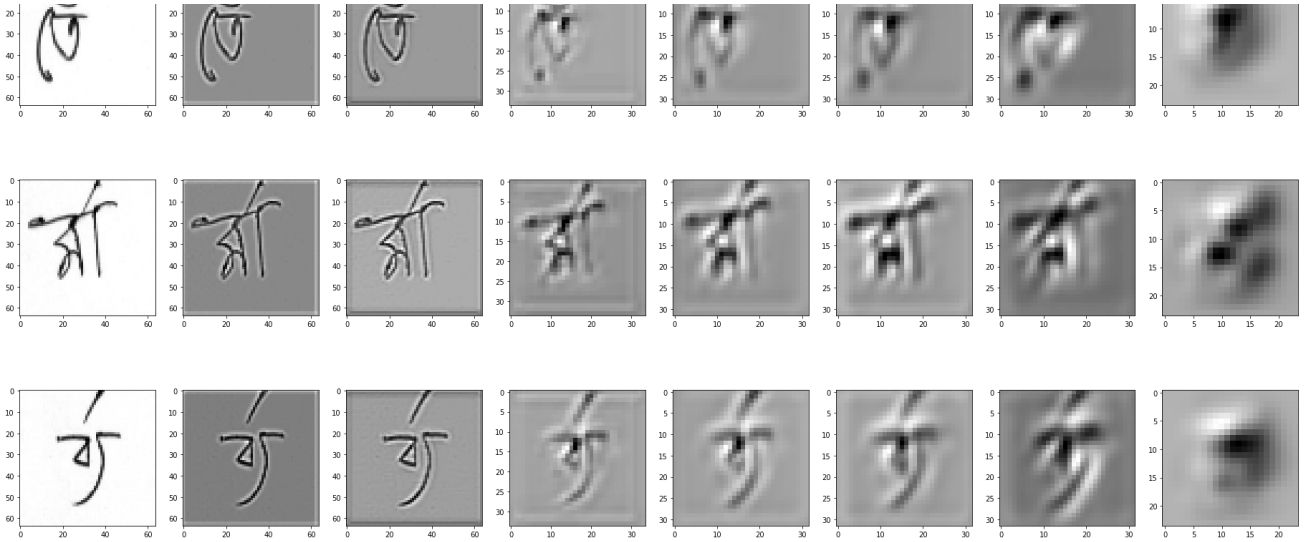
In [21]:

```
# Now create the output from only the first convolutional layer (using out "explore" model), second, an
d third
print("First examine twelve CORRECTLY classified examples")
pred_explore = []
for i in range (num_conv):
    pred_explore.append(model_explore[i].predict(X_correct))
    print("Have processed images through convolutional layer" ,i, "of the neural network. Shape is",
pred_explore[i].shape)
```

In [22]:

```
X_test = X_correct
# In this section we reverse the convolution layer to recreate a picture
num_img = X_test.shape[0]

# display the original and re-created images
fig, ax = plt.subplots(nrows=num_img, ncols=num_conv+1, figsize=(32, 64))

print("\n original image on the left followed by successive convolution layer recreations \n")

for i in range(num_img) : #
    col = 0
    # original image
    ax[i,col].imshow(X_test[i,:,:,0], cmap="gray")
    col += 1

    for j in range(num_conv) :
        patch = pred_explore[j].shape[1]
        # Recreate image from combination of this and all of the prior layers of filters, multiplied by
the output of this layer
        tot_border = int((cumulative_border[j] + border_siz[j]))
        scratch = np.zeros((patch + 2 * tot_border,patch + 2 * tot_border)) # The size of output plus
 borders
        for x in range(tot_border, patch+tot_border, 1) :
            for y in range(tot_border, patch+tot_border, 1) :
                for this_filter in range(num_filt[j]) :
                    # Find the color patch attributable to this Filter multiplied by the output value i
t generated
                    this_color_patch = np.copy(filter_patches[j][:,:,this_filter])
                    this_color_patch *= (pred_explore[j][i,x-tot_border,y-tot_border,this_filter])
                    scratch[x-tot_border:x+tot_border+1, y-tot_border:y+tot_border+1] += np.copy(this
_color_patch)
        size = scratch.shape[0]
        # PANv12f6 the border we choose not to display (because it's blank due to padding="same")
        #      may be too big when things scale down due to strides or max pooling...
        if scale_size[j] == 1 :
            tot_border = tot_border
        else:
            tot_border = int((tot_border / scale_size[j]) + 1)
        ax[i,col].imshow(scratch[tot_border:size-tot_border, tot_border:size-tot_border], cmap="gray"
)
#        ax[i,col].imshow(scratch, cmap="gray")   # here is if we want to show the full reconstruction
        col += 1

plt.show()
```



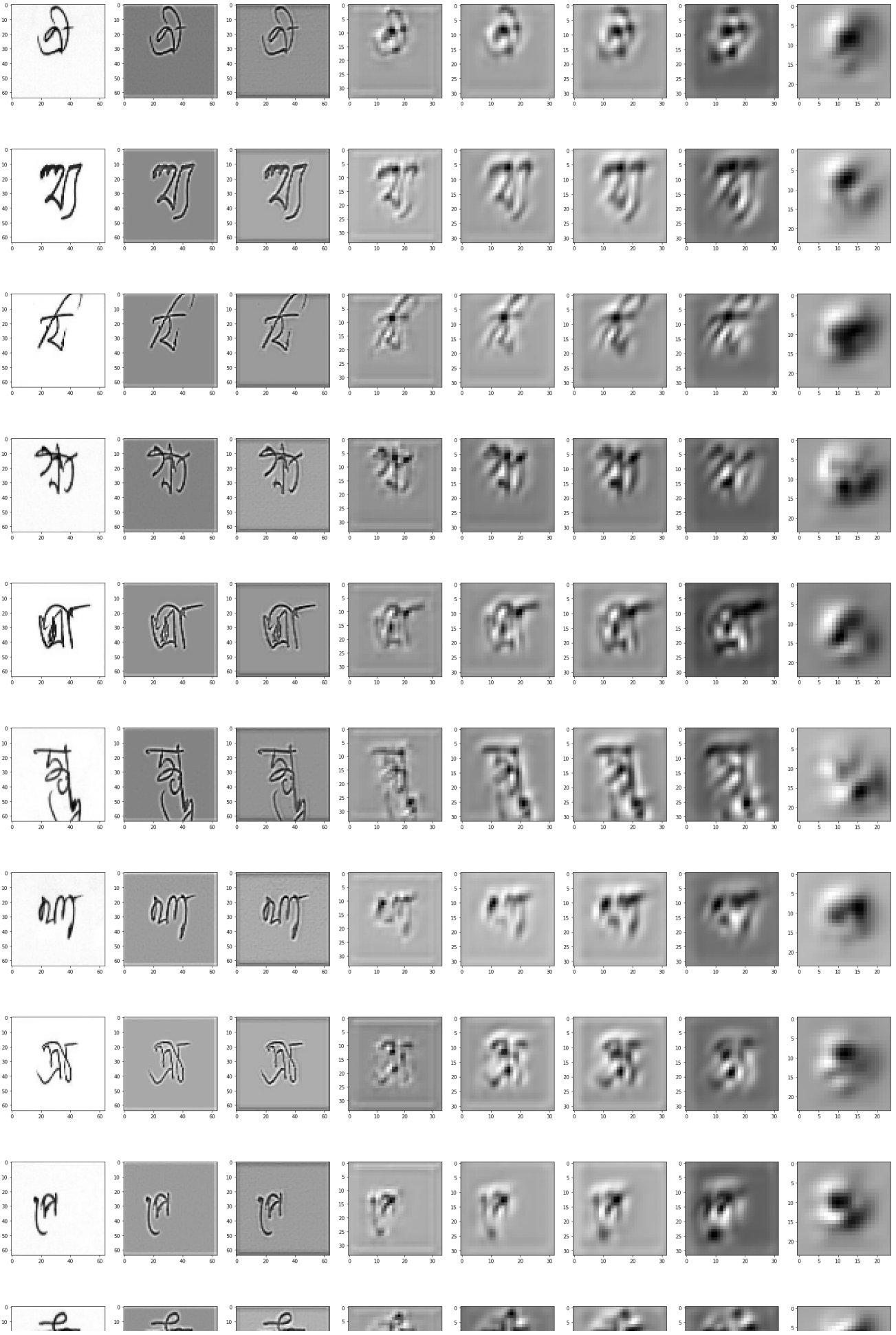


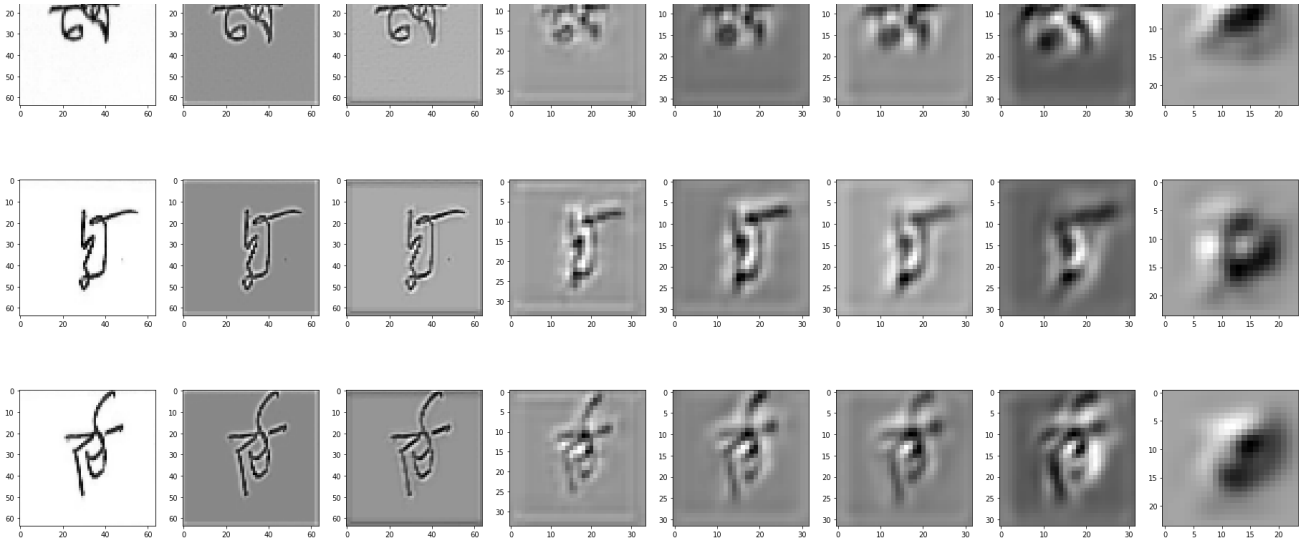
In [23]:

```
# Now create the output from only the first convolutional layer (using out "explore" model), second, an
d third
print("First examine twelve CORRECTLY classified examples")
pred_explore = []
for i in range (num_conv):
    pred_explore.append(model_explore[i].predict(X_incorrect))
    print("Have processed images through convolutional layer" ,i, "of the neural network. Shape is",
pred_explore[i].shape)
```

In [24]:

```
X_test = X_incorrect
# In this section we reverse the convolution layer to recreate a picture
num_img = X_test.shape[0]

# display the original and re-created images
fig, ax = plt.subplots(nrows=num_img, ncols=num_conv+1, figsize=(32, 64))

print("\n original image on the left followed by successive convolution layer recreations \n")

for i in range(num_img) : #
    col = 0
    # original image
    ax[i,col].imshow(X_test[i,:,:,0], cmap="gray")
    col += 1

    for j in range(num_conv) :
        patch = pred_explore[j].shape[1]
        # Recreate image from combination of this and all of the prior layers of filters, multiplied by
the output of this layer
        tot_border = int((cumulative_border[j] + border_siz[j]))
        scratch = np.zeros((patch + 2 * tot_border,patch + 2 * tot_border)) # The size of output plus
 borders
        for x in range(tot_border, patch+tot_border, 1) :
            for y in range(tot_border, patch+tot_border, 1) :
                for this_filter in range(num_filt[j]) :
                    # Find the color patch attributable to this Filter multiplied by the output value i
t generated
                    this_color_patch = np.copy(filter_patches[j][:,:,this_filter])
                    this_color_patch *= (pred_explore[j][i,x-tot_border,y-tot_border,this_filter])
                    scratch[x-tot_border:x+tot_border+1, y-tot_border:y+tot_border+1] += np.copy(this
_color_patch)
        size = scratch.shape[0]
        # PANv12f6 the border we choose not to display (because it's blank due to padding="same")
        #      may be too big when things scale down due to strides or max pooling...
        if scale_size[j] == 1 :
            tot_border = tot_border
        else:
            tot_border = int((tot_border / scale_size[j]) + 1)
        ax[i,col].imshow(scratch[tot_border:size-tot_border, tot_border:size-tot_border], cmap="gray"
)
#         ax[i,col].imshow(scratch, cmap="gray")   # here is if we want to show the full reconstruction
        col += 1

plt.show()
```





What can we conclude?

Note: Re-evaluated in light of fixes through version PANv12f

If you examine the eight (8) columns above, we see the original 64×64 input in the leftmost column, and moving left to right, we see a recreation of the input based on the output of consecutive convolution layers (Conv2D layers) and dense layers. Specifically:

- Column 1 - Original Input
- Columns 2 though 8 - Recreation of input based on output of Conv2D layer

In the case of the above, these recreations are done without any adjustment for biasing, normalization, or even max pooling. The only adjustment made for max pooling is the amount the displayed image is "zoomed in" so that it retains the original size of the input for comparison. For example; notice the image degredation when the first max pooling goes from 64×64 to 32×32 (between columns 3 and 4, or equivalently, between the 2nd and 3rd Conv2D). There is a HUGE degredation before the last convolution layer - perhaps due to MaxPooling from 32×32 down to 16×16, perhaps due to the 5×5 filter size, or perhaps something else?

Big problem with the last Conv2D layer

Is this because it's closest to the flatten and dense layers (morphed away from the original image becuase of this) or simply due to one-too-many MaxPoolings (leaving it at a 16×16 size from the original 64×64) or the increase in filter size from prior Conv2D at 3×3 to final Conv2D at 5×5??? More investigation is needed

Impact of Borders

In an earlier version of this kernel, I indicated there may be a problem with borders, but improvements to the "recreation of the original input" algorithm have shown me that this is not the case. I therefore retract the suggestion that borders need to be added to pad the data.

Impact of Resolution

Looking at the set of twelve "all correct" and twelve "all incorrect" examples, we see the impact of resolution gets worse and worse with each layer of convolution. This is a natural impact of the initial scaling of images to 64×64, the MaxPooling layers, as well as the strides parameter (not used here). It looks as though the "all correct" examples lose some information but enough remains to identify the feature. In the "all incorrect" some important detail features are entirely lost by the third convolution (column 4) which occurs after the first "MaxPooling" layer.

Possible solutions would be to reduce the use of MaxPooling, and also possibly keep the starting image at a higher resolution. Naturally, these both add to precious resource usage for this contest.

Examining the Filters Themselves

It is also worthwhile to examine the filters themselves. In all convolutional layers there seem to be no "duplicate" filters (no filters in the same convolutional layer that look identical), however, in the later convolutional layers, some look very similar to one another. There may be a method to perform "pruning" and use fewer filters.

An that's it for the Mind Reader

The above diagrams show the original test image, and then, in subsequent pairs of images, shows the output of the first second and 3rd convolutional layer, each followed by a reconstruction of the original image, based on that output.

An "expert" can clearly see if important information has been lost, what was lost, and even what layer within the neural network it was lost.

Please support by upvoting

If you feel this is an important area for further research, please upvote this notebook. This will help bring attention to seeing these or similar functions get incorporated within the Keras framework, and wherever it seems it will do the most good for future researchers. Perhaps others have come to the same conclusions and are already working similar efforts; in which case I am also happy to help where I can. I am a full time teacher, and as such, I can only devote spare time to this effort. Nevertheless, please feel free to reach out to me in this regard.

In [25]:
```
# This structure will be needed to create the contest output
preds_dict = {
    'grapheme_root': [],
    'vowel_diacritic': [],
    'consonant_diacritic': []
}
```

In [26]:
```
# Clean up data to prep for contest entry
del train_df
del X_correct, X_incorrect, X_train
del X_train
del Y_train_root, Y_train_vowel, Y_train_consonant
del model_explore, model_explore2, model_explore3
gc.collect()
```

```
---------------------------------------------------------------------------
NameError
<ipython-input-26-7ed845fa2eb4>    <module>
      2 del
      3 del          ,            ,
----> 4 del
      5 del             ,              ,
      6 del              ,               ,

NameError
```

In [27]:

```
# PANv00 Test on all 4 parquets (to generate output and get a LB score).
components = ['consonant_diacritic', 'grapheme_root', 'vowel_diacritic']
target=[] # model predictions placeholder
row_id=[] # row_id place holder
for i in range(4):
    df_test_img = pd.read_parquet('/kaggle/input/bengaliai-cv19/test_image_data_{}.parquet'.format(i
))
    df_test_img.set_index('image_id', inplace=True)

    X_test = resize(df_test_img, need_progress_bar=False)/255
    X_test = X_test.values.reshape(-1, IMG_SIZE, IMG_SIZE, N_CHANNELS)

    preds = model.predict(X_test)

    for i, p in enumerate(preds_dict):
        preds_dict[p] = np.argmax(preds[i], axis=1)

    for k,id in enumerate(df_test_img.index.values):
        for i,comp in enumerate(components):
            id_sample=id+'_'+comp
            row_id.append(id_sample)
            target.append(preds_dict[comp][k])
    del df_test_img
    del X_test
    gc.collect()

df_sample = pd.DataFrame(
    {
        'row_id': row_id,
        'target':target
    },
    columns = ['row_id','target']
)
df_sample.to_csv('submission.csv',index=False)
df_sample.head()
```

Out[27]:

	row_id	target
0	Test_0_consonant_diacritic	0
1	Test_0_grapheme_root	3
2	Test_0_vowel_diacritic	0
3	Test_1_consonant_diacritic	0
4	Test_1_grapheme_root	93