====Method 1: Simple CNN (AlexNet)====

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

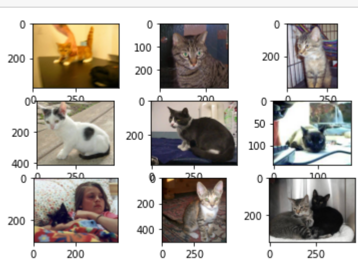
In [ ]:
```
ls
```

```
gdrive/  sample_data/
```

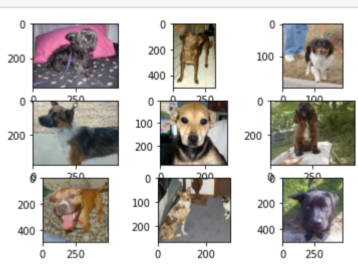
In [ ]:
```
import os, shutil

TRAINDIR = 'gdrive/MyDrive/ee_628/proj/train/'
cat_folder = 'cat/'
dog_folder = 'dog/'
```

Before, moving further, let us take a look at the images. This will give us an idea of what kind of data we are working with.

In [ ]:
```
from matplotlib.image import imread
import matplotlib.pyplot as plt

for i in range(9):
  plt.subplot(330 + 1 + i)
  filename = TRAINDIR+cat_folder + 'cat.' + str(i) + '.jpg'
  image = imread(filename)
  plt.imshow(image)
plt.show()
```



In [ ]:
```
for i in range(9):
  plt.subplot(330 + 1 + i)
  filename = TRAINDIR+dog_folder + 'dog.' + str(i) + '.jpg'
  image = imread(filename)
  plt.imshow(image)
plt.show()
```



There are two things to note from the above sample of images:

1) The images are all of different sizes and aspect ratios and need to be fitted to standard size.

2) All images seem to be at least bigger than 150X150 pixels and therefore we can attempt to resize them all to this size.

Below, we will first load in the data using ImageDataGenerator class from Keras.

In [ ]:
```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rotation_range = 10,
                             shear_range=0.2,
                             rescale=1./255,
                             validation_split=0.25)

IMG_H = 150
IMG_W = 150
```

In [ ]:
```
train_generator = datagen.flow_from_directory(TRAINDIR,
                                              target_size=(IMG_H, IMG_W),
                                              batch_size=100,
                                              class_mode='binary',
                                              subset='training')
```

```
Found 18750 images belonging to 2 classes.
```

In [ ]:
```
val_generator = datagen.flow_from_directory(TRAINDIR,
                                            target_size=(IMG_H, IMG_W),
                                            batch_size=100,
                                            class_mode='binary',
                                            subset='validation')
```

```
Found 6250 images belonging to 2 classes.
```

In [ ]:
```
test_generator = datagen.flow_from_directory('gdrive/MyDrive/ee_628/proj/', classes=['test1'],
                                             target_size=(IMG_H, IMG_W))
```

```
Found 12500 images belonging to 1 classes.
```

NOTE: As mentioned earlier, we do not really need the test_generator as we are not working with it. We have still loaded a generator for it because for the purposes of this method we generated some predictions and a csv file for the test data.

Let us build the model now:

In [ ]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization
```

In [ ]:
```
model1 = Sequential()
model1.add(Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(IMG_H, IMG_W, 3)))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))
model1.add(Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding='same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))
model1.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same'))
model1.add(BatchNormalization())
model1.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(3,3), strides=(2,2)))
model1.add(Flatten())
model1.add(Dense(4096, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(4096, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(1,activation='sigmoid'))
```

In [ ]:
```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 35, 35, 96)        34944
_________________________________________________________________
batch_normalization (BatchNo (None, 35, 35, 96)        384
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 17, 17, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       24832
_________________________________________________________________
batch_normalization_1 (Batch (None, 17, 17, 256)       1024
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 384)         885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 384)         1536
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 256)         884992
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              9441280
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 4097
=================================================================
Total params: 28,060,545
Trainable params: 28,058,561
Non-trainable params: 1,984
_________________________________________________________________
```

In [ ]:
```
model1.compile(loss='binary_crossentropy',
               optimizer = 'adam',
               metrics=['accuracy'])
```

In [ ]:
```
history1 = model1.fit_generator(train_generator, validation_data=val_generator, epochs=20)
```

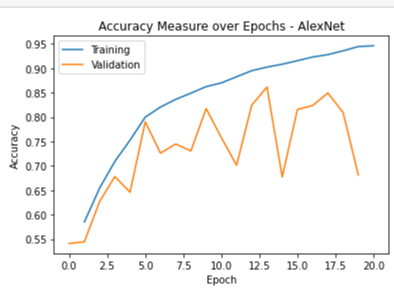
```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
```

```
Epoch 1/20
188/188 [==============================] - 8627s 46s/step - loss: 3.5865 - accuracy: 0.5531 - val_loss: 0.7119 - val_accuracy: 0.5411
Epoch 2/20
188/188 [==============================] - 727s 4s/step - loss: 0.6435 - accuracy: 0.6435 - val_loss: 0.6898 - val_accuracy: 0.5445
Epoch 3/20
188/188 [==============================] - 720s 4s/step - loss: 0.5779 - accuracy: 0.6972 - val_loss: 0.6349 - val_accuracy: 0.6269
Epoch 4/20
188/188 [==============================] - 727s 4s/step - loss: 0.5177 - accuracy: 0.7502 - val_loss: 0.6323 - val_accuracy: 0.6781
Epoch 5/20
188/188 [==============================] - 724s 4s/step - loss: 0.4486 - accuracy: 0.7964 - val_loss: 0.6258 - val_accuracy: 0.6461
Epoch 6/20
188/188 [==============================] - 724s 4s/step - loss: 0.4100 - accuracy: 0.8170 - val_loss: 0.4374 - val_accuracy: 0.7904
Epoch 7/20
188/188 [==============================] - 727s 4s/step - loss: 0.3689 - accuracy: 0.8383 - val_loss: 0.5439 - val_accuracy: 0.7262
Epoch 8/20
188/188 [==============================] - 727s 4s/step - loss: 0.3701 - accuracy: 0.8391 - val_loss: 0.5171 - val_accuracy: 0.7450
Epoch 9/20
188/188 [==============================] - 722s 4s/step - loss: 0.3049 - accuracy: 0.8702 - val_loss: 0.5137 - val_accuracy: 0.7306
Epoch 10/20
188/188 [==============================] - 724s 4s/step - loss: 0.3040 - accuracy: 0.8690 - val_loss: 0.4034 - val_accuracy: 0.8176
Epoch 11/20
188/188 [==============================] - 724s 4s/step - loss: 0.2703 - accuracy: 0.8850 - val_loss: 0.4714 - val_accuracy: 0.7579
Epoch 12/20
188/188 [==============================] - 717s 4s/step - loss: 0.2502 - accuracy: 0.8954 - val_loss: 0.6592 - val_accuracy: 0.7014
Epoch 13/20
188/188 [==============================] - 716s 4s/step - loss: 0.2273 - accuracy: 0.9056 - val_loss: 0.4076 - val_accuracy: 0.8246
Epoch 14/20
188/188 [==============================] - 714s 4s/step - loss: 0.2103 - accuracy: 0.9170 - val_loss: 0.3207 - val_accuracy: 0.8616
Epoch 15/20
188/188 [==============================] - 715s 4s/step - loss: 0.2080 - accuracy: 0.9190 - val_loss: 0.6800 - val_accuracy: 0.6773
Epoch 16/20
188/188 [==============================] - 724s 4s/step - loss: 0.1884 - accuracy: 0.9287 - val_loss: 0.4928 - val_accuracy: 0.8157
Epoch 17/20
188/188 [==============================] - 722s 4s/step - loss: 0.1764 - accuracy: 0.9295 - val_loss: 0.4269 - val_accuracy: 0.8237
Epoch 18/20
188/188 [==============================] - 723s 4s/step - loss: 0.1569 - accuracy: 0.9400 - val_loss: 0.4084 - val_accuracy: 0.8494
Epoch 19/20
188/188 [==============================] - 718s 4s/step - loss: 0.1355 - accuracy: 0.9480 - val_loss: 0.6208 - val_accuracy: 0.8094
Epoch 20/20
188/188 [==============================] - 712s 4s/step - loss: 0.1354 - accuracy: 0.9506 - val_loss: 1.0475 - val_accuracy: 0.6814
```

In [ ]:
```
history1.history
```

Out[ ]:
```
{'accuracy': [0.5851200222969055,
  0.6545066833496094,
  0.7090666890144348,
  0.7527999877929688,
  0.800000011920929,
  0.8206400275230408,
  0.8363199830055237,
  0.849120020866394,
  0.8624533414840698,
  0.8701333403587341,
  0.8827199935913086,
  0.8951466679573059,
  0.9025066494941711,
  0.9083733558654785,
  0.9156266450881958,
  0.923146665096283,
  0.9279999732971191,
  0.9358400106430054,
  0.9445866942405701,
  0.9462933540344238],
 'loss': [1.3889658451080322,
  0.6312090754508972,
  0.5641945004463196,
  0.5093401074409485,
  0.4435144364833832,
  0.40450748801231384,
  0.37481480836868286,
  0.3507314622402191,
  0.318170428276062,
  0.30447256565093994,
  0.2773495018482208,
  0.2543911337852478,
  0.23700104653835297,
  0.2273014634847641,
  0.21460409462451935,
  0.19635514914989471,
  0.18171706795692444,
  0.16720017790794373,
  0.14417661726474762,
  0.1427137702703476],
 'val_accuracy': [0.5411199927330017,
  0.54448002576828,
  0.6268799901008606,
  0.6780800223350525,
  0.6460800170898438,
  0.7904000282287598,
  0.7262399792671204,
  0.7449600100517273,
  0.7305600047111511,
  0.8176000118255615,
  0.7579200267791748,
  0.7014399766921997,
  0.8246399760246277,
  0.8615999817848206,
  0.6772800087928772,
  0.8156800270080566,
  0.8236799836158752,
  0.8494399785995483,
  0.809440016746521,
  0.681439995765686],
 'val_loss': [0.7118508815765381,
  0.6897774934768677,
  0.6349456906318665,
  0.632263720035553,
  0.6258352994918823,
  0.4374096989631653,
  0.5438859462738037,
  0.5171272158622742,
  0.5136682391166687,
  0.4033946394920349,
  0.4714355170726776,
  0.6591507792472839,
  0.407554566860199,
  0.32073676586151123,
  0.6800304651260376,
  0.4928198754787445,
  0.4269232749938965,
  0.40840908885002136,
  0.6208070516586304,
  1.0475376844406128]}
```

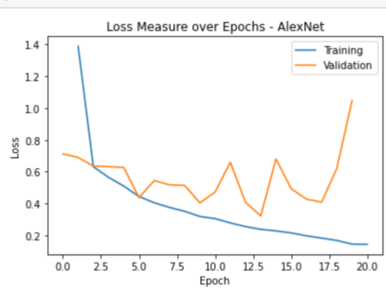
In [ ]:
```
epochs = [i for i in range(1,21)]

plt.plot(epochs, history1.history['accuracy'], history1.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Accuracy Measure over Epochs - AlexNet')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```



As we can see above, the training accuracy keeps on increasing, but we cannot keep increasing the epochs along with it. We see that the validation accuracy is fluctuating quite a bit. However, based on the current plot, we can make the deduction that the validation accuracy is still averaging around 0.75. If we run more epochs we would be able to make a clearer deduction and see where it is that the validation accuracy starts to converge. At that point we should stop the training of the neural net as it would lead to overfitting.

Currently, we could have more epochs but due to time constraints, we have kept 20 epochs (which took over 6 hours!)

In [ ]:
```
plt.plot(epochs, history1.history['loss'], history1.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Loss Measure over Epochs - AlexNet')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



We see in the plot above also that the loss fluctuates quite a bit for validation. It starts to creep up after the 17th epoch and signifies that we may already be starting to overfit too.

In [ ]:
```
model1.evaluate_generator(train_generator)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1877: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  warnings.warn('`Model.evaluate_generator` is deprecated and '
```

Out[ ]:
```
[0.8376135230064392, 0.7143466472625732]
```

In [ ]:
```
model1.metrics_names
```

Out[ ]:
```
['loss', 'accuracy']
```

In [ ]:
```
model1.evaluate_generator(val_generator)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1877: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  warnings.warn('`Model.evaluate_generator` is deprecated and '
```

Out[ ]:
```
[1.0505142211914062, 0.6852800250053406]
```

As we see above, the evaluation of the trained AlexNet architecture on the training and validation data has the following accuracy and loss measures.

	Accuracy %	Loss
Training	71.4	0.8376
Validation	68.5	1.0505

Below, I have worked to generate the csv file for predictions on the test set. The csv file generated can be used to make submission on the Kaggle 'Dogs vs Cats' competition.

In [ ]:
```
preds_alexnet = model1.predict(test_generator)
```

In [ ]:
```
type(preds_alexnet)
```

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
preds_alexnet
```

Out[ ]:
```
array([[0.6375611 ],
       [0.6368668 ],
       [0.06549668],
       ...,
       [0.7978575 ],
       [0.16250959],
       [0.00222257]], dtype=float32)
```

In [ ]:
```
class_preds_alexnet = model1.predict_classes(test_generator)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

In [ ]:
```
class_preds_alexnet
```

Out[ ]:
```
array([[0],
       [0],
       [0],
       ...,
       [1],
       [0],
       [1]], dtype=int32)
```

In [ ]:
```
preds_rounded_alexnet = preds_alexnet
```

In [ ]:
```
preds_rounded_alexnet[preds_rounded_alexnet <= 0.5] = 0
preds_rounded_alexnet[preds_rounded_alexnet > 0.5] = 1
```

In [ ]:
```
preds_rounded_alexnet
```

Out[ ]:
```
array([[1.],
       [1.],
       [0.],
       ...,
       [1.],
       [0.],
       [0.]], dtype=float32)
```

In [ ]:
```
preds_lst = [int(i) for i in preds_rounded_alexnet]
```

In [ ]:
```
len(preds_lst)
```

Out[ ]:
```
12500
```

In [ ]:
```
ids_lst = [i for i in range(1,12501)]
```

In [ ]:
```
len(ids_lst)
```

Out[ ]:
```
12500
```

In [ ]:
```
import pandas as pd

dct = {'id': ids_lst, 'label': preds_lst}
ans = pd.DataFrame(dct)
```

In [ ]:
```
ans.head()
```

Out[ ]:

	id	label
0	1	1
1	2	1
2	3	0
3	4	0
4	5	1

In [ ]:
```
ans.to_csv('alexnet_labels.csv', index=False)
```

In [ ]:
```
!cp alexnet_labels.csv "gdrive/MyDrive/ee_628/proj/"
```

When we used the predict_classes method it appears that the classes assigned a '1' to cats and '0' to dogs and we want it the other way for submission. Therefore for this list, we shall attempt to swap the values for each index in the array.

In [ ]:
```
indices_one = class_preds_alexnet == 1
indices_zero = class_preds_alexnet == 0
class_preds_alexnet[indices_one] = 0
class_preds_alexnet[indices_zero] = 1
```

In [ ]:
```
class_preds_alexnet
```

Out[ ]:
```
array([[1],
       [1],
       [1],
       ...,
       [0],
       [1],
       [0]], dtype=int32)
```

Sure enough, we can see by comparing it with earlier value of this array that this is the opposite.

In [ ]:
```
preds_classmethod_list = [int(i) for i in class_preds_alexnet]
```

In [ ]:
```
len(preds_classmethod_list)
```

Out[ ]:
```
12500
```

In [ ]:
```
dct_2 = {'id': ids_lst, 'label': preds_classmethod_list}
ans_2 = pd.DataFrame(dct_2)
```

In [ ]:
```
ans_2.head()
```

Out[ ]:

	id	label
0	1	1
1	2	1
2	3	1
3	4	0
4	5	1

In [ ]:
```
ans_2.to_csv('alexnet_labels_usingclassmethod.csv', index=False)
```

In [ ]:
```
!cp alexnet_labels_usingclassmethod.csv "gdrive/MyDrive/ee_628/proj/"
```