Spring 5-2019

Study of alpha mangostin as a chemoprotective agent for breast cancer via activation of the p53 pathway

Vanessa Van Oost
Olivet Nazarene University, vanessavanoost@sbcglobal.net

STUDY OF ALPHA MANGOSTIN AS A CHEMOPROTECTIVE AGENT FOR BREAST CANCER VIA ACTIVATION OF THE P53 PATHWAY

By

Vanessa V. Van Oost

Honors Scholarship Project

Submitted to the Faculty of

Olivet Nazarene University

for partial fulfillment of the requirements for

GRADUATION WITH UNIVERSITY HONORS

February 2019

BACHELOR OF SCIENCE

in

Biology

Scholarship Project Advisor	Signature	Date
Honors Council Chair	Signature	Date
Honors Council Member	Signature	Date

ACKNOWLEDGEMENTS

I would like to acknowledge the Olivet Nazarene University Honors Program as well as the Department of Biological Sciences for the support needed to complete this honors thesis. First of all, I would like to thank Dr. Gregory Long, my research mentor, for his ideas, patience, support, and help along the way. Furthermore, I have received incredible help from Dr. Sharda and Dr. Himes in the biology department as well. Without the support from these three professors, I could not have completed this research project. Not only did their expertise guide my research experience, but their encouragement helped me through the most difficult aspects of this experience.

I would like to thank the Hippenhammer Research Grant, the Pence Boyce Summer Research Program, and the Honors Program for the funds necessary to complete this project. The ONU biology department's resources including cell culture materials, microscopes, and other equipment have been of great assistance as well. I would also like to thank Evan Dexter for the development of the cell counting software that helped me incredibly with data collection. Finally, I would like to thank my family, friends, and Cohort 9 for support through the entire process.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Breast carcinoma is the most frequently diagnosed cancer among women and causes over 400,000 deaths each year worldwide. Current treatments such as chemotherapy are not selective for cancerous tissues but are destructive to normal tissues as well. This causes a range of side effects including pain, nausea, hair loss, weakness, and more. Inactivation of p53 is a very common mutation within human cancer cells. The ability to activate the p53 pathway which protects cells from tumor formation is lost in 50% of cancers. Due to the prevalence of this mutation, p53 is a uniquely valuable target for applied research. Alpha mangostin is an extract from a southeast Asian fruit, *Garcinia mangostana.* It has potential to be an effective p53 activator in which the small molecule disrupts the binding of p53 to MDM2, a negative regulator, inducing the p53 cascade which results in cell cycle arrest for low level stressors. This protects the cells from paclitaxel, a chemotherapy agent that only kills actively dividing cells. Here, we hypothesized that alpha mangostin protects wild-type, but not p53 (-/-), MCF10A breast cancer cells from the chemotherapeutic agent paclitaxel. When MCF10A wild-type cells were cotreated with alpha mangostin and paclitaxel, alpha mangostin exhibited a protective effect on the cells. However, when MCF10A P53 knockout cells were treated with alpha mangostin, cell viability decreased indicating a loss of protective effect in the p53 distressed cancer cells. These results further support treatments that target chemoprotection via p53 pathway in wild-type cells and the use of alpha mangostin warrants further study.

LITERATURE REVIEW

Breast Carcinoma & Treatments

The development of cancer is a multistep process that involves complex interactions between host and tumor tissue. The process involves oncogene activation as well as immunosuppression, leading to uncontrolled cell growth despite damaged DNA (Wang 2010). Genome instability contributes to cancer development due to mutations in DNA damage response pathways, which are mediated through the tumor suppressor p53 (Reinhardt & Schumacher, 2012).

Breast carcinoma is the most frequently diagnosed cancer among women and causes over 400,000 deaths yearly worldwide (Walerych, Napoli, Collavin, & Del Sal, 2012). Metastasis accounts for a large majority of the deaths that result from breast cancer, mostly to lymph nodes, lungs, and bones. The complexity of cancer leads to difficulties in treatment. Many of these difficulties are due to the nonselective nature of the treatment which cause cell toxicity in noncancerous cells. Treatment plans are dependent on the stage and characteristics of the cancer, as well as the age, menopausal status, and risk benefit analysis associated with each option. Based on a study in 2017, stage I and II patients most often receive breast conserving surgery. Stage III patients most often receive a mastectomy, radiation therapy, as well as chemotherapy with a five-year survival rate of 72%. Stage IV patients most often receive radiation therapy and or chemotherapy with a five-year survival rate of about 22% (American Cancer Society, 2017).

In order to delay the progression of breast cancer and increase the longevity of patients, less toxic yet effective chemotherapeutic agents are needed to limit the debilitating side effects while also improving outcomes (Shibata et al., 2011). These side effects include pain, lymphedema, musculoskeletal symptoms, bone loss and osteoporosis, heart problems, new cancers, blood clots, infertility, and loss of memory and cognitive function. (American Cancer Society, 2017).

Currently, conventional treatments such as radiation, chemotherapy, and surgery have not been entirely effective against the high incidence and low survival rates of breast cancer due to its complex nature (Moongkarndi et al., 2004). Research has established that combinations of drugs are more effective than one drug alone for the treatment of early-stage breast cancer. An example of such treatment is Trastuzumab, a monoclonal antibody that directly targets the human epidermal growth factor 2 (HER2) protein.

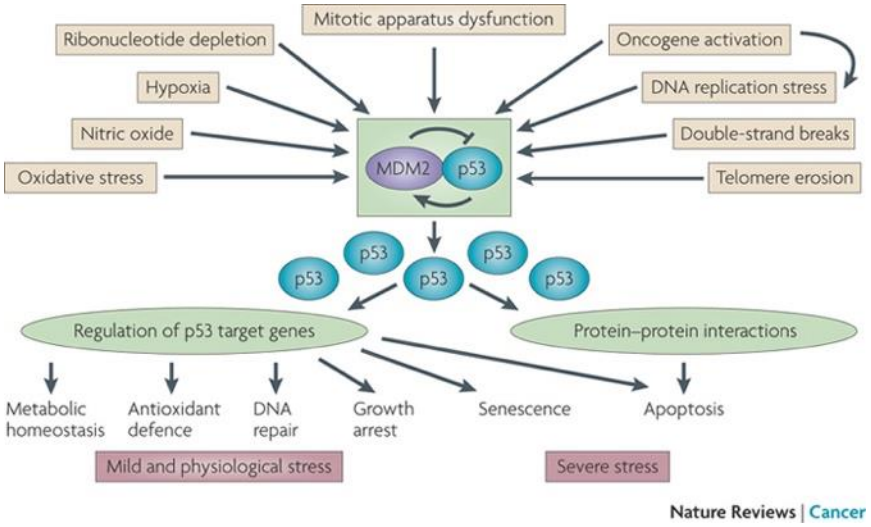

Figure 1. Various outcomes of the p53 pathway. Different causes, such as those of mild physiological stress or severe stress, interrupt the mdm2 and p53 binding and result in various outcomes such as DNA repair, growth arrest, apoptosis, and more (Vogelstein, Hughes, Kimmel, & Cancer, 2013).

When combined with chemotherapy, this treatment was found to reduce the risk of recurrence by 52% and death by 33% for patients who overproduce the growth promoting protein HER2/neu (American Cancer Society, 2017). This leads the scientific community to search for other potential therapeutic approaches including drug combinations to treating this malignancy and many other cancers that continue to evade conventional treatments.

P53 Gene & Cascade

The main function of p53 is to promote genetic stability and prevent the formation of tumors. When a cell is under stress, it induces cell death through apoptosis for severe stress or cell arrest and subsequent DNA repair for mild stress in order to prevent malignant growth. Under normal conditions, p53 levels are low and the binding to mouse double minute 2 homolog (MDM2) targets it to the proteasome for rapid degradation and inhibition of its transcriptional activity (Burgess et al., 2016).

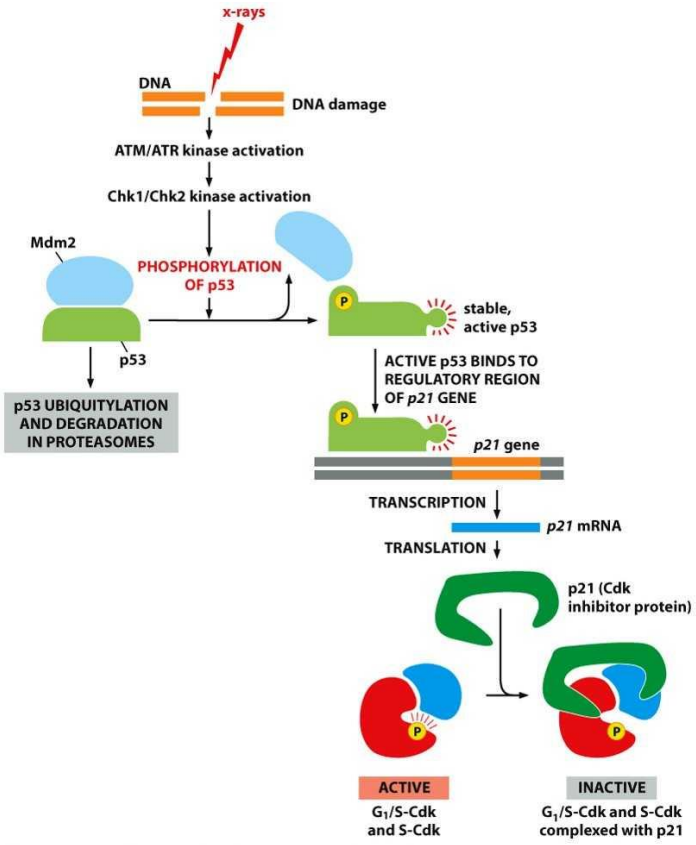

Figure 17-62 Molecular Biology of the Cell 6e (© Garland Science 2015)

Figure 2. The p53 Cascade. This figure is a simple representation of a cell's response to stress that result in DNA damage. The first step is phosphorylation of p53 to affect its binding to mdm2. This process also includes a key player in the process, p21, which is a Cdk inhibitor. The end result is an inactivation of the G1.S-Cdk and D-Cdk complex with p21. (Vogelstein, Hughes, Kimmel, & Cancer, 2013).

When stress occurs, binding to the regulatory protein MDM2 is disrupted due to P53 phosphorylation, which leads to p53 accumulation and subsequent transcription of numerous genes, including the gene that encodes the cyclin-dependent kinase inhibitor (CKI) protein p21. P21 binds and inactivates G_1/S-Cdk complexes, arresting the cell in G_1 for DNA repair.

P53 Mutation & Treatments

Inactivation of p53 is a very common feature of human cancer cells (Lane, Cheok, & Lain, 2010). About 50% of adult cancer has p53 inactivated (by mutation or deletion) while the other 50% have suppressed wild-type p53 function (Choong, Yang, Lee, & Lane, 2009). On average, p53 is mutated in 20% of tumors in breast cancer. Though the frequency of mutation is lower in breast cancer cells, p53 inactivation has been seen in some breast cancers without a mutation. The pathway has been shown to be affected by alterations in upstream regulatory proteins and downstream p53-induced proteins (Gasco, Shami, & Crook, 2002). In breast cancer, this mutation is associated with a more aggressive disease and worse overall survival according to several studies (Gasco et al., 2002).

The ability to activate the p53 pathway which protects cells from tumor formation is lost in cells with p53 mutations. Most of these mutations occur as a result of a substitution of single amino acids in the central region of the p53 protein, which causes many variants (Walerych et al., 2012). Indeed, rapid malignant cell growth which leads to many different cancer types often involves a defective p53 gene, which is the transcriptional activator that works to suppress tumors in normal tissues (Muller &

Vousden, 2014). In breast cancer, mutant p53 is involved in many processes associated with cancer development such as early tumorigenesis, tumor growth and development, and metastasis (Walerych et al., 2012). In clinical practice, molecular pathological analysis of the tumors of the structure and expression and constituents of the p53 pathway is likely to have value in diagnosis, in prognostic assessment and, ultimately, in treatment of breast cancer (Gasco et al., 2002).

Chemotherapy and Mechanism of Chemoprotection

Chemotherapy induces many adverse effects in patients because of normal cell toxicity, resulting at least in part from p53 activation and apoptosis induction in normal proliferating cells/tissues, such as bone marrow, lymphoid organs, hair follicles, and epithelium lining of the small intestine (Wang & Sun, 2010). An important aspect of chemotherapy is it kills actively dividing cells. In particular, paclitaxel inhibits microtubule function, which kills cells as they enter mitosis (Blagosklonny, 2002). Microtubules are essential to the process of mitosis as they separate chromosomes to opposite sides of the cell during anaphase. When paclitaxel inhibits the ability for the chromosomes to be separated during the division process, the cell is inactivated and eventually is killed. Therefore, wild-type cells treated with the p53 activator are arrested in G1 and do not enter into mitosis and therefore the chemotherapy selectively kills p53-deficient cancer cells. This mechanism can be experimentally controlled with the use of p53 activators to arrest p53 wild-type cells and protect against the harmful effects of chemotherapeutic agents on noncancerous cells.

P53 Activator Current Research

Due to the prevalence of this mutation, p53 is a uniquely valuable target for applied research (Vogelstein et al., 2013). Therefore, much research has gone into both therapeutic strategies to restore mutant to wild-type p53 and pretreatment of cancer cells with p53 activators, that arrest noncancerous p53-normal proliferating cells without impacting the cell cycle of cells with a p53 mutation, thus allowing for selective killing of cancerous cells.

Current research on this type of treatment has led to the discovery of small molecules that directly or indirectly activate p53. Some p53 activation has been achieved in the clinic. The most advanced of these are the p53 MDM2 interaction inhibitors. The first class of small molecule mdm2 inhibitors discovered was nutlin-3a, which binds to the hydrophobic cleft in the N-terminus of mdm2, preventing its association with p53 and initiating the cascade. Since this discovery of nutlin-3a, many more related compounds have been tested, some of which have now made it to the preclinical stage. This stage will better assess the biological effects and toxicity of the treatment to patients (Burgess et al., 2016). As research continues and understanding of p53 response increases, development will continue allowing for powerful drug combinations that may increase the selectivity and safety of chemotherapy, by selective protection of normal cells and tissue (Lane et al., 2010).

Alpha Mangostin as a Chemoprotectant

Alpha mangostin is a p53 activator that is isolated from the carp of the *Garcinia mangostana* (Mangosteen fruit), which is native to Thailand and traditionally used for antioxidant, antitumoral, antiallergic, anti-inflammatory, antibacterial, and antiviral

medicinal purposes (Pedraza-Chaverri, Cárdenas-Rodríguez, Orozco-Ibarra, & Pérez-Rojas, 2008). This extract is known to inhibit the binding of p53 to MDM2, a negative regulator of p53.

In one study in 2011, alpha mangostin was used to reduce tumor growth and lymph node metastasis in an immunocompetent xenograft model of metastatic mammary cancer with a p53 mutation. The study showed that treatment with 20 mg/kg/day alpha mangostin resulted in prolonged survival rates and increased inhibition of tumor growth and lymph node metastasis (Shibata et al., 2011). This reveals that this extract at high concentrations can potentially be a successful treatment for p53 mutated cancer types. Meanwhile, at lower concentrations, alpha mangostin has the potential to act as a chemoprotectant to wild-type cells. One study tested the chemoprotection of alpha mangostin on wild-type BHK cells. The results supported the hypothesis that the alpha mangostin can be used to protect cells from the cytotoxic effects of chemotherapy (Wojciechowski, 2017). However, the effect on breast cancer cell lines at low concentrations is not known. Here, this research seeks to determine whether or not the alpha mangostin pretreatment would be an effective strategy for chemoprotection of wild-type cells by testing whether or not the cancer cells are also protected. If the data indicate that the cancerous cells are not protected, this p53 activator could potentially be a successful pretreatment before chemotherapy for selective cancer killing. I hypothesize that alpha mangostin, a p53- dependent chemoprotectant, protects wild-type cells, but not those with a p53 mutation from the chemotherapeutic agent paclitaxel.

MATERIALS AND METHODS

Culturing of MCF10A Cells

Cell culture protocol was based on the ATCC© Thawing, Propagating, and Cryopreserving Protocol ("Thawing, Propagating, and Cryopreserving Protocol: MCF10A-JSB Breast epithelium," 2012). MCF10A p53 Wild-type and p53 knockout (-/-) human breast cancer cells were cultured in Dulbecco's Modified Eagle's Medium supplemented with Cholera Toxin from *V. cholera*, Insulin solution, Epidermal Growth Factor, 50 uM Hydrocortisone Solution, and Horse Serum at 37^0 C. PBS was used to rinse the wells before lifting. Trypsin-EDTA solution was used to lift cells.

Determining the Toxicity of Alpha Mangostin and Paclitaxel

Cells were treated at various levels of alpha mangostin and paclitaxel to create a dose response curve and toxicity curve and determine workable concentrations for the dual treatment experiments. For the dual treatment, cells were treated with various concentrations of alpha mangostin, ranging from 0-.25 uM, for 24 hours followed by a 24 hour dual treatment of alpha mangostin and 15 nM paclitaxel.

Differentiating between Viable, Apoptotic, and Necrotic Cells

Cells were stained with Hoechst, YO-PRO 1, and Propidium Iodide to differentiate between viable, apoptotic, and necrotic cells. Data was collected on cell viability using differential fluorescent staining. The differences were shown through fluorescent microscopy with Hoechst, Propidium Iodide, and YO-PRO-1 stains. The Hoechst stain only stains normal healthy cells. The next two, Propidium Iodide and YO-PRO-1, only stains necrotic and apoptotic cells, respectively. Through these three stains,

I will be able to only evaluate the attached cells for comparison. Quantitatively, the cells stained with Hoechst stain were used for the results section. However, the two other stains are shown in the figures with photos. Representative pictures at 10x were taken blindly by a professor in order to eliminate bias and counted using the fluorescent cell counting program.

Development and Use of Cell Counting Program

The program used to count the cells was developed by Evan Dexter using Microsoft Visual Studio in C# language. The code for this program is shown in Appendix I. It works by use of an algorithm that examines the color values of the pictures' pixels. Pixels with high values of the color being counted are flagged as potential cell locations. Various user-controlled parameters are then used to refine the number of cells from the initial list. Changing the parameters makes it possible to count the number of cells for different types of cells and conditions. The resulting count is displayed to the user as both a sum total and a marker on each cell location (Dexter 2019).

RESULTS

Data shown in the results section come from representative photos taken at 100x on a fluorescent microscope and analyzed using a double-blind method. Typical methods such as the cell hemocytometer were not able to be used due to difficulties lifting the cells without killing them. Therefore, developed a program that would count representative photos to have an idea of cell viability in each of the treatment wells (appendix). Viable cells were counted as indicated by the blue Hoechst stain.

Concentration ranges for toxicity curves and dose response curves were based upon previous research using these treatments with baby hamster kidney cells (Wojciechowski, 2017).

Single Treatment Results for Wild-Type Cells (p53 Pathway intact)

We first wanted to determine the effect of the chemotherapeutic drug, paclitaxel, on the MCF10A wildtype cells. This procedure allowed for the determination of a concentration that would be suitable for dual treatment with alpha mangostin.

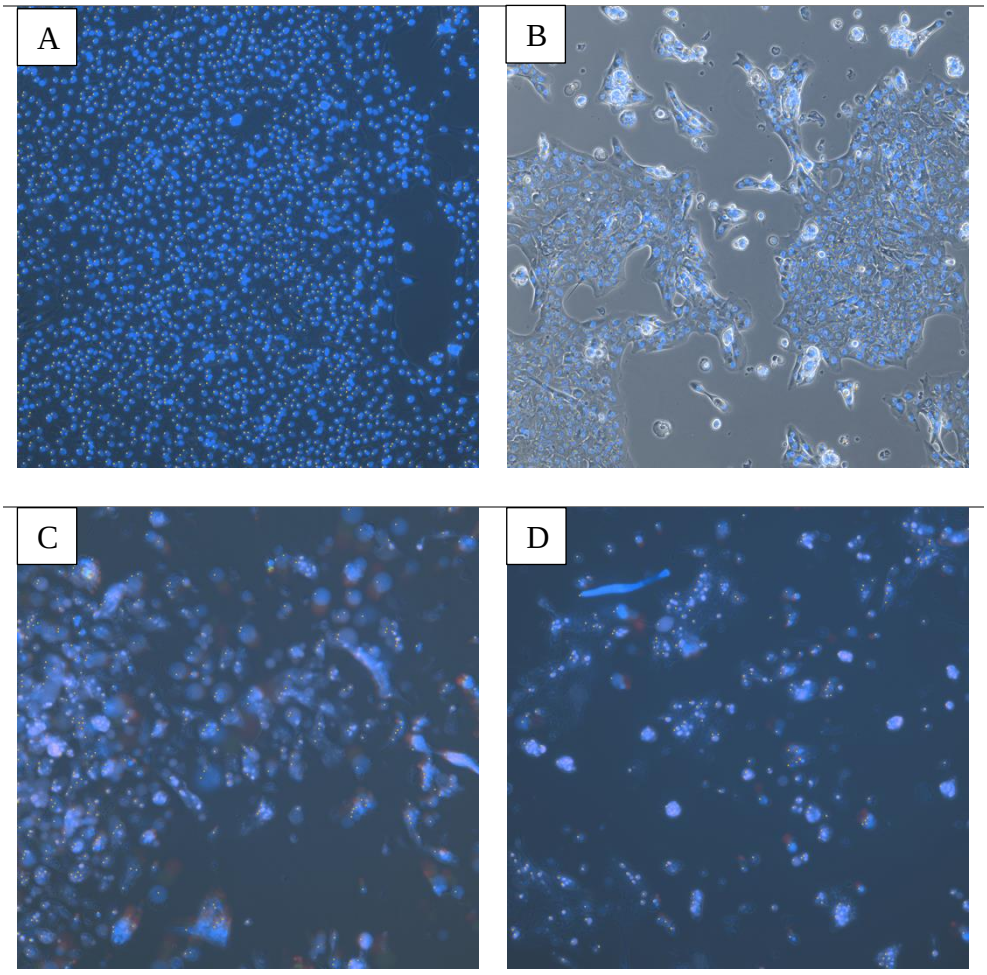

Figure 3. Fluorescent Microscopy of MCF10A Wild-Type Cells after treatment with Paclitaxel (A is the control, B is 10 nM, C is 25 nM, and D is 35 nM). The orange markers on the photos indicate that the cell was

counted using the fluorescent cell counting program. Increase in concentration led to a decrease in live cell count.

Increasing concentrations of alpha mangostin caused a decrease in the number of cells in the well (Figures 3 & 4). There were not many floating dead cells in the wells which indicate that the decrease in cell number was likely due to cell cycle arrest. The control well had a variable number of cells in each well, as indicated by the large error bars in Figure 4. If experimental error did not have an effect on the results of the control well, then very low concentrations of paclitaxel had a mitogenic effect on the wild-type MCF10A cell line. The chosen concentration of paclitaxel for the dual treatment was 15 nM as higher concentration did not give a sufficient number of cells to allow for variable cell counts with lower standard error.

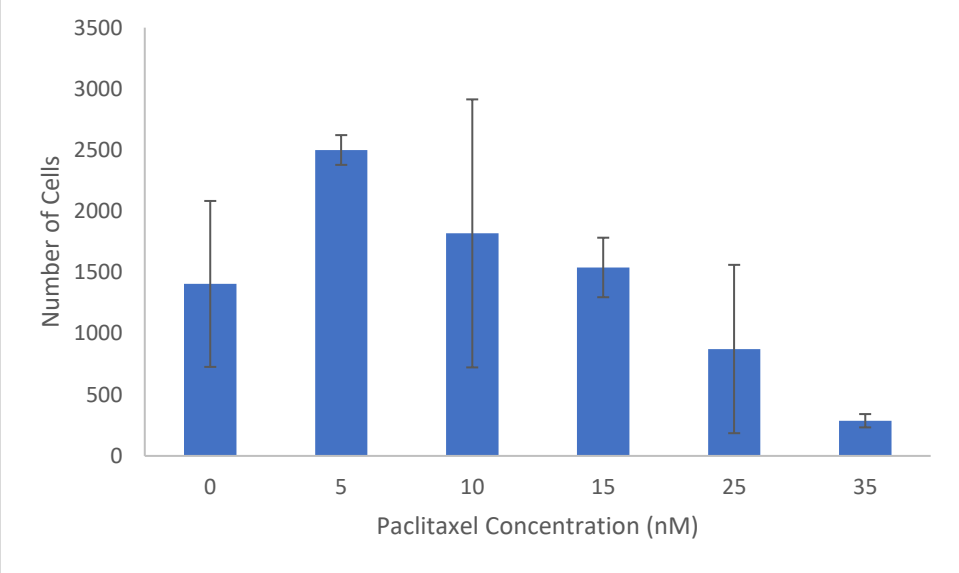

Figure 4. Wild-Type MCF10A Paclitaxel Toxicity Curve. Wild-Type MCF10A Paclitaxel Toxicity Curve. Cells were treated with concentrations ranging from 0-35 nM for 24 hours and the cell count per field of view was recorded per well. The 5, 10, and 15 nM concentration of Paclitaxel had a mitogenic effect on the cells with toxicity beginning at 25 nM.

We next sought to determine the effect of alpha mangostin on cell viability for the wild-type MCF10A cells. This experiment was also done in order to determine an acceptable range for the dual treatment for this cell line with the chemotherapeutic agent, paclitaxel.

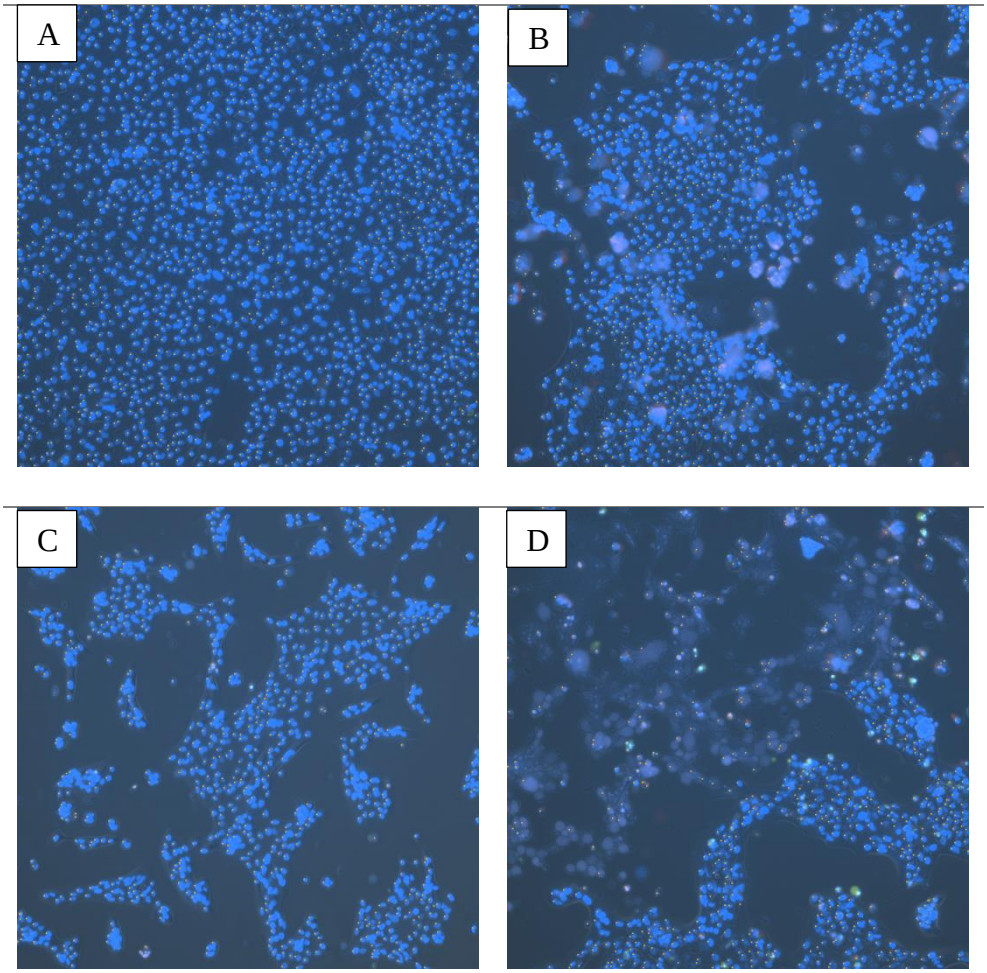

Figure 5. Fluorescent Microscopy of MCF10A Wild-Type Cells after treatment with Alpha Mangostin (A is the control, B is 0.05 µM, C is 0.15 µM, D is 0.25 µM). The orange markers on the photos indicate that the cell was counted using the fluorescent cell counting program. Increase in concentration led to a decrease in live cell count.

The MCF10A wild-type cells responded in a dose dependent manner (Figure 5 & 6). In addition to the control, varying concentrations ranging from 0.05 to 0.25 were used because it gave a suitable range that had variable cell counts.

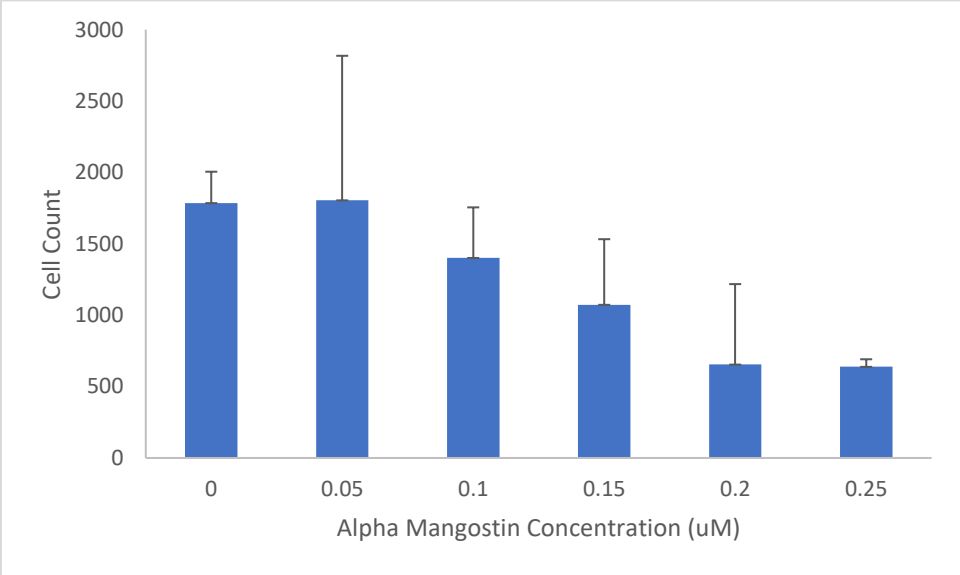

Figure 6. Wild-Type MCF10A Alpha Mangostin Dose Response Curve. Cells were treated with varying concentrations of alpha mangostin for 24 hours and counted using the fluorescent cell counting program.

Dual Treatment Results for Wild-Type Cells (p53 Pathway intact)

The final experiment for the wild-type cells was the dual treatment with alpha mangostin for 24 hours followed by a combination of alpha mangostin and paclitaxel for another 24 hours.

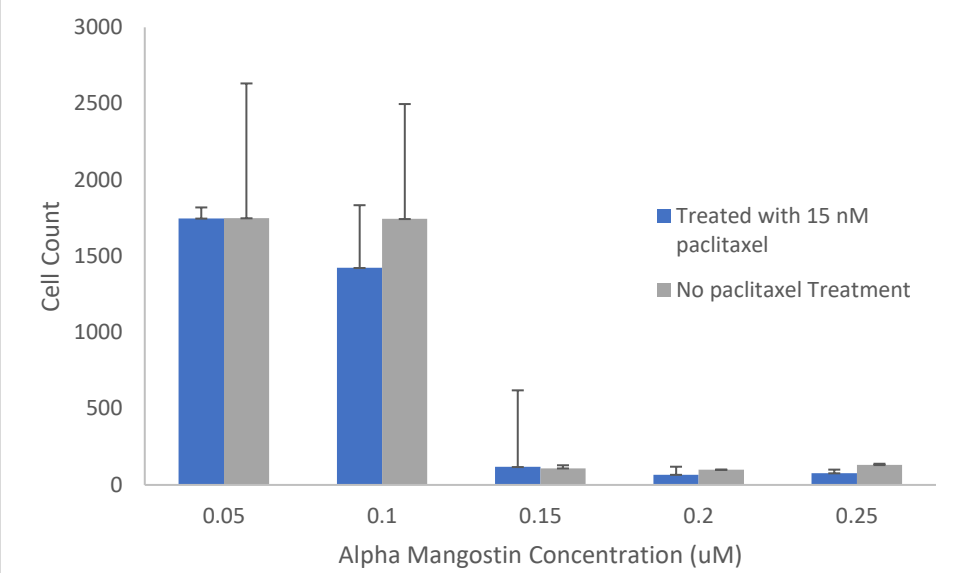

Figure 7. Wild-Type MCF10A Alpha Mangostin & Paclitaxel Dual Treatment. Cells were treated with varying concentrations of alpha mangostin for 24 hours and counted using the fluorescent cell counting program.

Cells treated with alpha mangostin alone and those cotreated with paclitaxel yielded similar cell counts (Figure 7). While these results indicate a negligible effect of alpha mangostin as a chemoprotectant, we also observed a wide variety of experimental error as will be discussed further below.

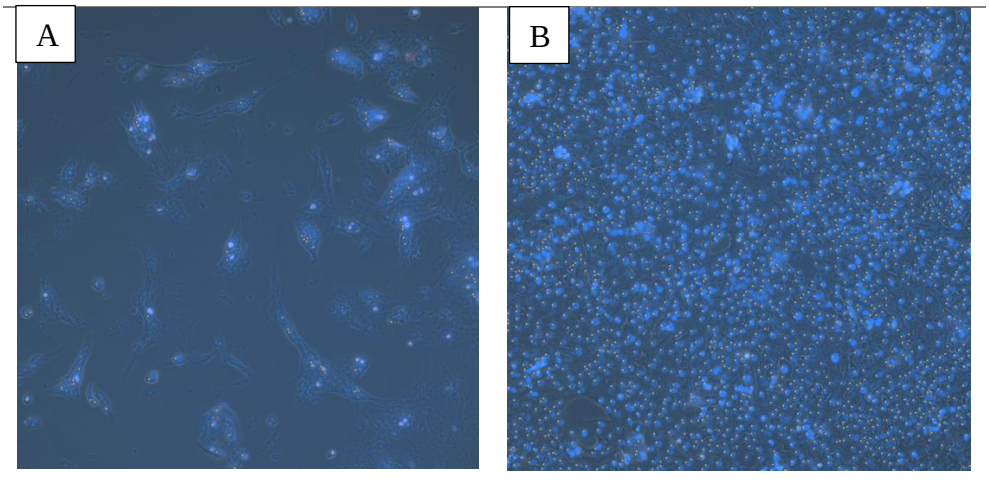

Figure 8. Fluorescent Microscopy of Wild-Type MCF10A Paclitaxel Treatment vs Dual Treatment with Alpha Mangostin. A is the paclitaxel

only treatment and B is the alpha mangostin paclitaxel dual treatment. The orange markers on the photos indicate that the cell was counted using the fluorescent cell counting program.

Single Treatment Results for MCF10A p53 (-/-) Cells

P53 is known to be an important regulator of cell cycle arrest and as noted above is often absent in numerous cancers. We therefore wanted to determine the role of a chemoprotectant such as alpha mangostin on p53 knockout cells (Figures 9 & 10). The concentration range used was slightly larger than that used for the wild-type cells to gain a better understanding of the effect of the concentration on the treatment.

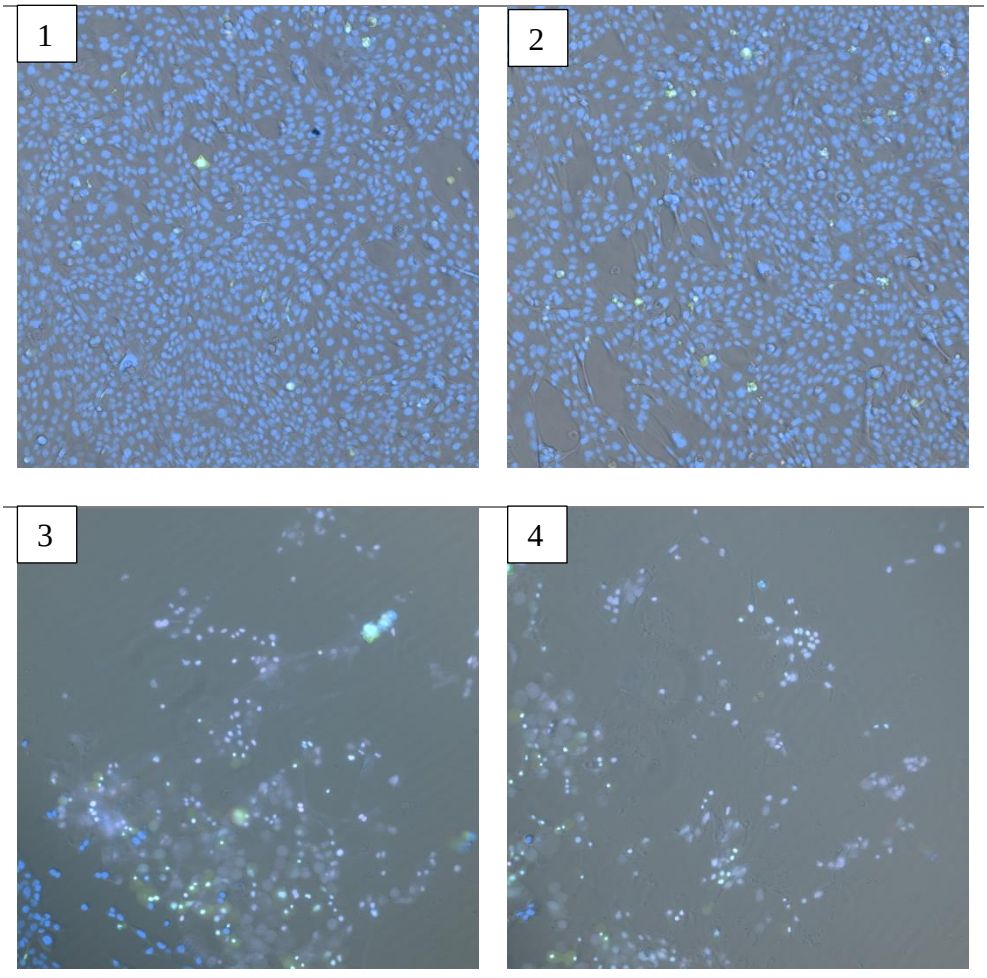

Figure 9. Fluorescent Microscopy of MCF10A p53 (-/-) Cells after treatment with Alpha Mangostin (1 is the control, 2 is 0.1 µM, 3 is 0.5 µM, 4 is 1 M µM). Blue cells indicate live cells stained with Hoechst, and green cells indicate apoptotic cells, and red cells indicate necrotic cells. Increasing concentrations led to a decrease in live cell count.

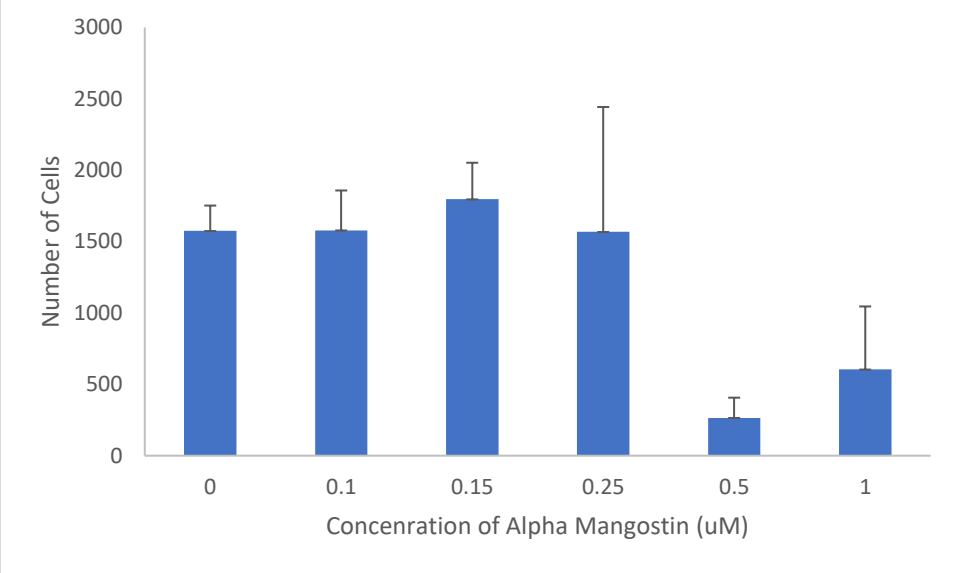

Figure 10. MCF10A p53 (-/-) Alpha Mangostin Dose Response Curve. Cells were treated with varying concentrations of alpha mangostin for 24 hours and counted using the fluorescent cell counting program.

In addition to quantitative data, the wells were qualitatively assessed through a blinded observation to determine the approximate cell viability (Figure 11). It was important to perform this study for this experiment because the obtained counts for Figures 9 & 10 were not particularly representative. This is because photos were taken at the edges of the wells since the middle of the well had limited consistency in all treatments. Viability was greatly diminished at concentration of 0.25 or greater, which is in contrast to what was observed with MCF10A wild-type cells.

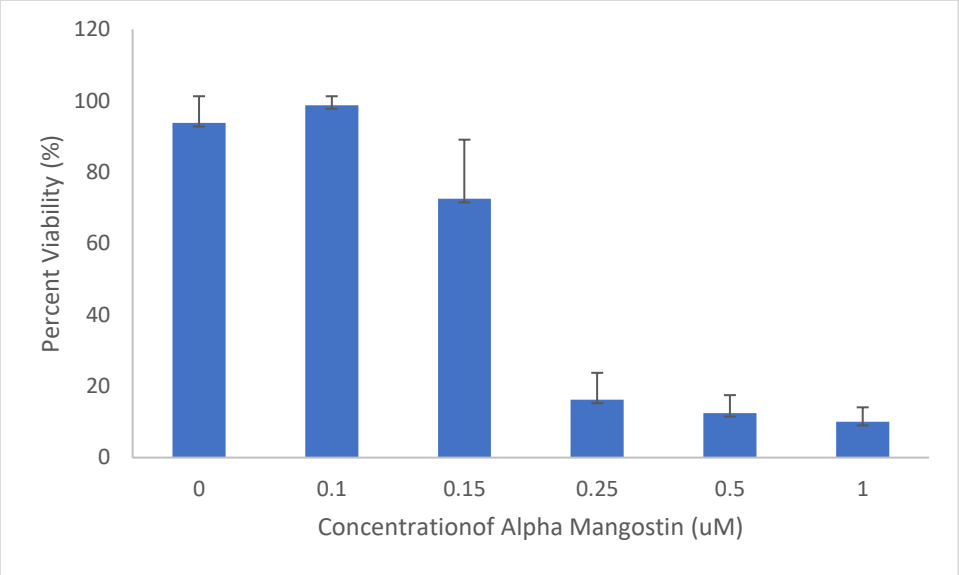

Figure 11. Qualitative Observation of Percent Viability for Increasing Treatments of Alpha Mangostin on MCF10A p53 (-/-) cells. Observations of this percentage were estimated using a blind study method.

DISCUSSION

Through induction of p53-dependent G2 arrest, pretreatment can prevent cell death caused by microtubule inhibiting drugs such as paclitaxel. This would allow for selective killing of p53 mutant cancer cells (Blagosklonny, 2002). In order to assess the chemoprotective abilities of alpha mangostin, we performed several experiments on MCF10A wildtype and p53 (-/-) breast cancer cells. When working with this cell line, a lot of troubleshooting and method alterations went into the process of creating the experiments. This is important to note because the major time commitment for this troubleshooting did not allow for duplication of the experiments. In order to fully draw conclusions from this research, one must duplicate the data in order to gain confidence in the results presented. Also, it is important to note that the key findings presented in figure 11 came from a double blinded study to avoid bias.

To begin, the MCF10A wildtype cells were treated with both alpha mangostin and paclitaxel separately to create a toxicity and dose response curve in order to determine a workable range of concentrations for alpha mangostin and an optimum concentration of paclitaxel for the dual treatment. As shown in figure 4, a 15 nM concentration of paclitaxel had higher confidence, shown by smaller error bars, and had a significant fraction of less cells than the 5 nM treatment. The mitogenic effect of lower doses in contrast to the low number of cells in the control well may be a result of experimental error. This concentration has the potential to give room for variation of cell number when pretreated with alpha mangostin. Thus, the 15 nM concentration was chosen to be the fixed paclitaxel concentration for the dual treatment.

Figure 6 shows that alpha mangostin responds in a dose dependent manner. These concentrations were shown to be effective to be used for the dual treatments. Therefore, the next experiment included a dual treatment in which cells were treated with concentrations of 0.00 μM to 0.25 μM of alpha mangostin on day 1 and treated with both alpha mangostin and 15 nM paclitaxel on day 2 (Figure 7). The results of this dual treatment show similar cell counts for cultures treated with and without paclitaxel, indicating that alpha mangostin is able to effectively protect cells from the normally cytotoxic effects of paclitaxel. This conclusion is based on the paclitaxel toxicity curve that shows that adding each subsequent concentration of paclitaxel should decrease cell number. However, the cells treated with alpha mangostin and no paclitaxel and the cells that were dual treated had very similar values. Overall, the wells that were pretreated with alpha mangostin had a much larger live cell count than the well that received no

alpha mangostin pretreatment before it was treated with paclitaxel (Figure 8). Because the wild-type cells are also representative of typical human breast cells with wild-type p53, these experiments indicate that human breast cells will be protected from paclitaxel by alpha mangostin.

The second section of results evaluates the effect of pretreatment of p53 knockout MCF10A cells to ensure that these cells are not protected by the alpha mangostin. If the alpha mangostin showed a similar protective effect, then it would not be a useful selective treatment for breast cancer. To explore the relationship between the knockout cells and the alpha mangostin, a dose response treatment, ranging from 0 to 1 µM alpha mangostin, was administered to knockout cells for 24 hours (Figure 10). Based on the mechanism of p53 activators, which includes inhibition of mdm2-p53 interaction in the normal pathway, cell count numbers were expected to be consistent in different concentrations of alpha mangostin. However, number of cells decreased with increasing alpha mangostin concentration. Further analysis revealed that the wells with higher concentrations of alpha mangostin had many dead cells which were floating and thus not recognized by the fluorescent microscope. These dead cells indicate that the low number of cells at high concentrations is not due to cell arrest, but rather from cell toxicity. The decrease in cell number is shown in Figures 9, 10 and 11. Figure 10 contains quantitative data that shows a relative decrease in cell number correlates to an increase in concentration. Figure 11 contains a qualitative observation of percent viability for increasing treatments of alpha mangostin on MCF10A p53 (-/-) cells. The percent confluence decreased as alpha mangostin increased. Overall, these data show that the

number of cells depends on the concentration of the alpha mangostin in a dose dependent manner.

Overall, the range of concentration of alpha mangostin between 0.1 µM to 0.25 µM showed a protective benefit to wild-type MCF10A breast cancer cells. Furthermore, the same concentration range decreased cell viability of the MCF10A p53 knockout cell line. This indicates that alpha mangostin has potential as a selective cancer treatment when paired with the chemotherapeutic drug paclitaxel. While these results are encouraging and suggest that alpha mangostin is a potential chemoprotectant for chemotherapy, it is important to acknowledge the limitations of this research. Due to difficulties and troubleshooting, only one trial for each of these various treatments was completed. Therefore, in order to fully validate this research, it must be replicated to eliminate any results which came from experimental error. Along with replication of this experiment, it would be beneficial to continue the knockout MCF10A p53 (-/-) study and perform a toxicity curve with paclitaxel in order to perform a dual treatment study. Furthermore, these experiments should be performed on several different cell lines to evaluate the effectiveness of alpha mangostin on different cell types and cancers. Further research is required to explore the promising pretreatment to chemotherapy, alpha mangostin.

Although many treatments exist for cancer, scientists are always looking for better and more effective options for patients. When evaluating treatment plans, patients must consider both the effectiveness of the treatment and the negative side effects of each treatment option. While chemotherapy has shown to be an effective and aggressive

treatment option, it is not selective for that tissue, and therefore has many harsh side effects including pain, lymphedema, musculoskeletal symptoms, bone loss and osteoporosis, heart problems, new cancer development, blood clots, infertility, and loss of memory and cognitive function (American Cancer Society, 2017). This research, as well as other studies, show that alpha mangostin has potential as a p53 activation pretreatment before chemotherapy to limit these side effects.

The p53 gene is a great potential target for cancer treatment because it is the most frequently altered gene in human cancers (Shibata et al., 2011). If alpha mangostin can be used as a tool for chemoprotection in a p53-dependent manner in breast cancer development, then it would enable doctors to treat patients with higher concentrations of chemotherapy without harming healthy tissue. It would also potentially have implications in treatments of other types of cancer, such as some uterine, ovarian, and lung cancers, which also have a high prevalence of p53 mutations. Alpha mangostin offers a more natural treatment option that can be easily translated to clinical use because it is already FDA approved. Overall, this study shows that alpha mangostin warrants further research to better understand its effectiveness as a pretreatment for chemotherapy.

BIBLIOGRAPHY

American Cancer Society. (2017). Breast Cancer Facts & Figures 2017-2018. *Breast Cancer Facts & Figures*, 1–44. https://doi.org/10.1007/s10549-012-2018-4.Mesothelin

Blagosklonny, M. V. (2002). Sequential activation and inactivation of G2 checkpoints for selective killing of p53-deficient cells by microtubule-active drugs. *Oncogene*, *21*(41), 6249–6254. https://doi.org/10.1038/sj.onc.1205793

Burgess, A., Chia, K. M., Haupt, S., Thomas, D., Haupt, Y., & Lim, E. (2016). Clinical Overview of MDM2/X-Targeted Therapies. *Frontiers in Oncology*, *6*(January), 1–7. https://doi.org/10.3389/fonc.2016.00007

Choong, M. L., Yang, H., Lee, M. A., & Lane, D. P. (2009). Specific activation of the p53 pathway by low dose actinomycin D: A new route to p53 based cyclotherapy. *Cell Cycle*, *8*(17), 2810–2818. https://doi.org/10.4161/cc.8.17.9503

Gasco, M., Shami, S., & Crook, T. (2002). The p53 pathway in breast cancer. *Breast Cancer Research*, *4*(2), 70–76. https://doi.org/10.1186/bcr426

Lane, D. P., Cheok, C. F., & Lain, S. (2010). p53-based Cancer Therapy. *Cold Spring Harbor Perspectives in Biology*, *2*(9), a001222--a001222. https://doi.org/10.1101/cshperspect.a001222

Moongkarndi, P., Kosem, N., Kaslungka, S., Luanratana, O., Pongpan, N., & Neungton, N. (2004). Antiproliferation, antioxidation and induction of apoptosis by Garcinia mangostana (mangosteen) on SKBR3 human breast cancer cell line. *Journal of Ethnopharmacology*, *90*(1), 161–166. https://doi.org/10.1016/j.jep.2003.09.048

Muller, P. A. J., & Vousden, K. H. (2014). Mutant p53 in cancer: New functions and therapeutic opportunities. *Cancer Cell*, *25*(3), 304–317. https://doi.org/10.1016/j.ccr.2014.01.021

Pedraza-Chaverri, J., Cárdenas-Rodríguez, N., Orozco-Ibarra, M., & Pérez-Rojas, J. M. (2008). Medicinal properties of mangosteen (Garcinia mangostana). *Food and Chemical Toxicology*, *46*(10), 3227–3239. https://doi.org/10.1016/j.fct.2008.07.024

Reinhardt, H. C., & Schumacher, B. (2012). The p53 network: Cellular and systemic DNA damage responses in aging and cancer. *Trends in Genetics*, *28*(3), 128–136. https://doi.org/10.1016/j.tig.2011.12.002

Shibata, M. A., Iinuma, M., Morimoto, J., Kurose, H., Akamatsu, K., Okuno, Y., … Otsuki, Y. (2011). α-Mangostin extracted from the pericarp of the mangosteen (Garcinia mangostana Linn) reduces tumor growth and lymph node metastasis in an immunocompetent xenograft model of metastatic mammary cancer carrying a p53 mutation. *BMC Medicine*, *9*, 1–18. https://doi.org/10.1186/1741-7015-9-69

Thawing, Propagating, and Cryopreserving Protocol: MCF10A-JSB Breast epithelium. (2012). *ATTC*, *1*, 1–27.

Vogelstein, B. B., Hughes, M. D. H., Kimmel, S., & Cancer, C. (2013). p53 : The Most Frequently Altered Gene in Human Cancers How Do We Know p53 Is a Tumor Suppressor Gene ?, 1–8.

Walerych, D., Napoli, M., Collavin, L., & Del Sal, G. (2012). The rebel angel: Mutant p53 as the driving oncogene in breast cancer. *Carcinogenesis*, *33*(11), 2007–2017.

https://doi.org/10.1093/carcin/bgs232
Wang, Z., & Sun, Y. (2010). Targeting p53 for Novel Anticancer Therapy. *Translational Oncology*, *3*(1), 1–12. https://doi.org/10.1593/tlo.09250
Wojciechowski, A. C. (2017). Using α-Mangostin from Garcinia mangostana to Block Cell Death Caused by Paclitaxel in Proliferating BHK Cells.

APPENDIX I

Fluorescent Cell Counting Software Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;

namespace Fluorescent_Cell_Counter
{
   public partial class Form1 : Form
   {
      public static Form1 me;

      #region Variables
      List<Bitmap> bitmaps = new List<Bitmap>();
      List<Bitmap> origBitmaps = new List<Bitmap>();
      Bitmap bitmap; //used for current one in analysis
      List<string> picNames = new List<string>();
      List<int[]> results = new List<int[]>();
      List<List<Point>> markers = new List<List<Point>>();
      List<List<Point>> badMarkers = new List<List<Point>>();
      List<List<string>> badMarkersExp = new List<List<string>>();
      List<Area> areas = new List<Area>();
      Area activeArea = null;
      bool defaultArea = false;
      int currentPic = 0;
      bool analyzing = false;
      bool selectingArea = false;
      bool mousePressed = false;
      int zoomLevel = 0;//0 is normal, 1 is one zoom in, -1 is one zoom out
      bool movingSettings = false;
      Point settingsOffset;

      int analysisMethod = 0; int CUTOFF = 0; int DIFFERENCE = 1;
      bool lookingForGreen = true; int GREEN = 0;
      bool lookingForRed = false; int RED = 1;
      bool lookingForBlue = false; int BLUE = 2;

      int colorMinVal = 200;
      int graySens = 20;
      int whiteSens = 150;
      double minCellDist = 12;
      double maxCellDist = 45;
      int minCellSize = 5;
      int maxCellSize = 30;
      int pixelSearchRange = 1;
      int differenceRange = 10;
      int colorDifferenceAmount = 30;
      int edgeBuffer = 3;
      int defaultAreaSize = 1400;
      double cellMult = 1;
      double volume = .5;
      int pixelScanVal = 2;
      #endregion

      public Form1()
      {
         me = this;
         InitializeComponent();
         timer1.Start();
         methodBox.SelectedIndex = 0;

         helpPanel.Parent = this;
         helpPanel.Location = new Point(230, 68);
      }

      private void timer1_Tick(object sender, EventArgs e)
      {
         if (movingSettings)
            advancedPanel.Location = new Point(Cursor.Position.X - settingsOffset.X, Cursor.Position.Y - settingsOffset.Y);
         if (analyzing || bitmaps.Count == 0 || advancedPanel.Visible == true)
            return;

         #region Area
         if (selectingArea && mousePressed)
         {
            if (MouseButtons.Equals(MouseButtons.Right))
            {
               selectingArea = false;
               Form1_MouseUp(this, new MouseEventArgs(System.Windows.Forms.MouseButtons.Left, 0, 0, 0, 0));
               return;
            }
            Area a = new Area(this.PointToClient(Cursor.Position), new Point(1, 1));
            activeArea = a;
```

```
    selectingArea = false;
    Form1_MouseUp(this, new MouseEventArgs(System.Windows.Forms.MouseButtons.Left, 0, 0, 0, 0));
    return;
}
if (activeArea != null)
{
    if (defaultArea)
    {
        Point apparentSize = new Point((int)(defaultAreaSize * Math.Pow(1.5, zoomLevel)), (int)(defaultAreaSize * Math.Pow(1.5, zoomLevel)));
        activeArea.moveArea(new Point(this.PointToClient(Cursor.Position).X - apparentSize.X, this.PointToClient(Cursor.Position).Y - apparentSize.Y), apparentSize);
    }
    else
    {
        activeArea.moveArea(activeArea.location, this.PointToClient(Cursor.Position), 0);
    }
    if (mousePressed)
    {
        if (MouseButtons.Equals(MouseButtons.Right))
        {
            selectingArea = false;
            activeArea.dispose();
            activeArea = null;
            defaultArea = false;
            Form1_MouseUp(this, new MouseEventArgs(System.Windows.Forms.MouseButtons.Left, 0, 0, 0, 0));
            return;
        }
        if (zoomLevel < 0)
            for (int i = zoomLevel; i < 0; i++)
            {
                activeArea.bitmapLoc = new Point((int)(activeArea.bitmapLoc.X * 1.5), (int)(activeArea.bitmapLoc.Y * 1.5));
                activeArea.bitmapSiz = new Point((int)(activeArea.bitmapSiz.X * 1.5), (int)(activeArea.bitmapSiz.Y * 1.5));
            }
        if (zoomLevel > 0)
            for (int i = zoomLevel; i > 0; i--)
            {
                activeArea.bitmapLoc = new Point((int)(activeArea.bitmapLoc.X * 2 / 3), (int)(activeArea.bitmapLoc.Y * 2 / 3));
                activeArea.bitmapSiz = new Point((int)(activeArea.bitmapSiz.X * 2/3), (int)(activeArea.bitmapSiz.Y * 2/3));
            }
        areas.Add(activeArea);
        activeArea = null;
        defaultArea = false;
        selectingArea = false;
        Form1_MouseUp(this, new MouseEventArgs(System.Windows.Forms.MouseButtons.Left, 0, 0, 0, 0));
    }
    return;
}
#endregion

bool found = false;
#region Regular Markers
foreach (Point m in markers.ElementAt(currentPic))
{
    Point loc = this.PointToClient(Cursor.Position);
    Point formLoc = loc;
    loc.Offset(-pictureBox.Location.X, -pictureBox.Location.Y);
    if (zoomLevel < 0)
        for (int i = zoomLevel; i < 0; i++)
        {
            loc = new Point((int)(loc.X * 1.5), (int)(loc.Y * 1.5));
        }
    if (zoomLevel > 0)
        for (int i = zoomLevel; i > 0; i--)
        {
            loc = new Point((int)(loc.X * 2 / 3), (int)(loc.Y * 2 / 3));
        }
    if (Math.Abs(m.X - loc.X) < 4 && Math.Abs(m.Y - loc.Y) < 4)
    {
        found = true;
        Point mForm = m;
        if (zoomLevel < 0)
            for (int i = zoomLevel; i < 0; i++)
            {
                mForm = new Point((int)(mForm.X * 2 / 3), (int)(mForm.Y * 2 / 3));
            }
        if (zoomLevel > 0)
            for (int i = zoomLevel; i > 0; i--)
            {
                mForm = new Point((int)(mForm.X * 1.5), (int)(mForm.Y * 1.5));
            }
        mForm.Offset(pictureBox.Location);
        if (highlight.Location != new Point(mForm.X - 4, mForm.Y - 4))
        {
            highlight.Location = new Point(mForm.X - 4, mForm.Y - 4);
            highlight.BringToFront();
        }
        highlight.Visible = true;
        highlight.Show();
        highlight.Refresh();
        if (mousePressed)
        {
            int[] i = results.ElementAt(currentPic);
            int c = bitmaps.ElementAt(currentPic).GetPixel(m.X, m.Y).ToArgb();
            if (c == Color.Red.ToArgb())
```

```
                i[GREEN]--;
            else if (c == Color.White.ToArgb())
                i[RED]--;
            else if (c == Color.Orange.ToArgb())
                i[BLUE]--;
            for (int x = -2; x <= 2; x++)
            {
                for (int y = -2; y <= 2; y++)
                {
                    if (m.X + x >= 0 && m.X + x < bitmaps.ElementAt(currentPic).Width && m.Y + y >= 0 && m.Y + y < bitmaps.ElementAt(currentPic).Height)
                    {
                        Color o = origBitmaps.ElementAt(currentPic).GetPixel(m.X + x, m.Y + y);
                        bitmaps.ElementAt(currentPic).SetPixel(m.X + x, m.Y + y, o);
                    }
                }
            }
            pictureBox.Image = bitmaps.ElementAt(currentPic);
            pictureBox.Refresh();
            markers.ElementAt(currentPic).Remove(m);
            results.Insert(currentPic, i);
            results.RemoveAt(currentPic + 1);
            updateResults();
            highlight.Visible = false;
            Form1_MouseUp(this, new MouseEventArgs(System.Windows.Forms.MouseButtons.Left, 0, 0, 0, 0));
        }
        break;
    }
  }
  #endregion

  if (!found)
    highlight.Visible = false;
}

public void workingAnimation()
{
  progressPic.Image.RotateFlip(RotateFlipType.Rotate90FlipNone);
  progressPic.Refresh();
}

public void analyze()
{
  if (bitmaps.Count == 0)
    return;
  analyzing = true;
  for (int i = 0; i < markers.Count; i++)
  {
    markers.Insert(i, new List<Point>());
    markers.RemoveAt(i + 1);
  }
  //
  for (int i = 0; i < badMarkers.Count; i++)
  {
    badMarkers.Insert(i, new List<Point>());
    badMarkers.RemoveAt(i + 1);
  }
  for (int i = 0; i < badMarkersExp.Count; i++)
  {
    badMarkersExp.Insert(i, new List<string>());
    badMarkersExp.RemoveAt(i + 1);
  }
  //
  currentPic = 0;
  resultBox.Text = "";
  for (int i = 0; i < bitmaps.Count; i++)
  {
    int numGreen = 0;
    int numRed = 0;
    int numBlue = 0;
    resultBox.Text += "Pic " + (i + 1) + " of " + bitmaps.Count + ":" + Environment.NewLine;
    if (lookingForGreen)
    {
      numGreen = maximaMethod(i, GREEN);
      resultBox.Text += "Green: " + numGreen + Environment.NewLine;
    }
    if (lookingForRed)
    {
      numRed = maximaMethod(i, RED);
      resultBox.Text += "Red: " + numRed + Environment.NewLine;
    }
    if (lookingForBlue)
    {
      numBlue = maximaMethod(i, BLUE);
      resultBox.Text += "Blue: " + numBlue + Environment.NewLine;
    }
    results.Insert(currentPic, new int[]{numGreen, numRed, numBlue});
    results.RemoveAt(currentPic + 1);

    pictureBox.Image = bitmaps.ElementAt(currentPic);
    pictureBox.Refresh();

    if (i != bitmaps.Count - 1)
      nextButton_Click(this, new EventArgs());
```

```
    }
    updateResults();
    analyzing = false;
}

public int maximaMethod(int num, int color)
{
    List<Point> maxima = new List<Point>();

    bitmap = bitmaps.ElementAt(num);
    if (areas.Count == 0)
    {
        for (int x = edgeBuffer; x < bitmap.Width - edgeBuffer; x += pixelScanVal)
        {
            if (Math.Abs(x % 50) < pixelScanVal)
                workingAnimation();
            for (int y = edgeBuffer; y < bitmap.Height - edgeBuffer; y += pixelScanVal)
            {
                if (analysisMethod == CUTOFF)
                {
                    if (pixelRGB(bitmap.GetPixel(x, y), color) >= colorMinVal && !isBadColor(bitmap.GetPixel(x, y), color))
                    {
                        maxima.Add(findMaxima(new Point(x, y), color));
                    }
                }
                else if (analysisMethod == DIFFERENCE)
                {
                    if (differentEnough(x, y, color) && !isBadColor(bitmap.GetPixel(x, y), color))
                    {
                        maxima.Add(findMaxima(new Point(x, y), color));
                    }
                }
            }
        }
    }
    else
    {
        foreach (Area a in areas)
        {
            for (int x = a.bitmapLoc.X; x < a.bitmapLoc.X + a.bitmapSiz.X; x += pixelScanVal)
            {
                if (Math.Abs(x % 50) < pixelScanVal)
                    workingAnimation();
                for (int y = a.bitmapLoc.Y; y < a.bitmapLoc.Y + a.bitmapSiz.Y; y += pixelScanVal)
                {
                    if (x < bitmap.Width && y < bitmap.Height)
                    {
                        if (analysisMethod == CUTOFF)
                        {
                            if (pixelRGB(bitmap.GetPixel(x, y), color) >= colorMinVal && !isBadColor(bitmap.GetPixel(x, y), color))
                            {
                                maxima.Add(findMaxima(new Point(x, y), color));
                            }
                        }
                        else if (analysisMethod == DIFFERENCE)
                        {
                            if (differentEnough(x, y, color) && !isBadColor(bitmap.GetPixel(x, y), color))
                            {
                                maxima.Add(findMaxima(new Point(x, y), color));
                            }
                        }
                    }
                }
            }
        }
    }

    return cleanMaximaList(maxima, num, color);
}

public int cleanMaximaList(List<Point> maxima, int num, int color)
{
    List<Point> maximaNoDup = new List<Point>();
    int count = 0;
    int pointCount = 0;
    foreach (Point p in maxima)
    {
        bool found = false;
        Point toAdd = new Point(-1, -1);
        if (pointCount % 50 == 0)
        {
            workingAnimation();
            Application.DoEvents();
        }
        for (int i = maximaNoDup.Count - 1; i >= 0; i--)
        {
            if ((p.X - maximaNoDup.ElementAt(i).X) > maxCellDist)
                break;
            if (p.X == maximaNoDup.ElementAt(i).X && p.Y == maximaNoDup.ElementAt(i).Y)
            {
                found = true;
                break;
            }
```

```
            if (Math.Sqrt((p.X - maximaNoDup.ElementAt(i).X) * (p.X - maximaNoDup.ElementAt(i).X) + (p.Y - maximaNoDup.ElementAt(i).Y) * (p.Y - maximaNoDup.ElementAt(i).Y)) < minCellDist)
            {
               found = true;
               toAdd = p;
               break;
            }
         }
         if (toAdd != new Point(-1, -1))
         {
            maximaNoDup.Add(toAdd);
            /*
            badMarkers.ElementAt(num).Add(toAdd);
            badMarkersExp.ElementAt(num).Add("Too close to other maxima");
            for (int x = -2; x <= 2; x++)
            {
               for (int y = -2; y <= 2; y++)
               {
                  if (p.X + x >= 0 && p.X + x < bitmaps.ElementAt(num).Width && p.Y + y >= 0 && p.Y + y < bitmaps.ElementAt(num).Height)
                  {
                     Color c = bitmaps.ElementAt(num).GetPixel(p.X + x, p.Y + y);
                     if (c != Color.Red && c != Color.White && c != Color.Orange)
                        bitmaps.ElementAt(num).SetPixel(p.X + x, p.Y + y, Color.Gray);
                  }
               }
            }
            */
            pointCount++;
            continue;
         }
         if (isBadColor(bitmap.GetPixel(p.X, p.Y), color))
         {
            /*
            badMarkers.ElementAt(num).Add(p);
            badMarkersExp.ElementAt(num).Add("Bad color");
            for (int x = -2; x <= 2; x++)
            {
               for (int y = -2; y <= 2; y++)
               {
                  if (p.X + x >= 0 && p.X + x < bitmaps.ElementAt(num).Width && p.Y + y >= 0 && p.Y + y < bitmaps.ElementAt(num).Height)
                  {
                     Color c = bitmaps.ElementAt(num).GetPixel(p.X + x, p.Y + y);
                     if (c != Color.Red && c != Color.White && c != Color.Orange)
                        bitmaps.ElementAt(num).SetPixel(p.X + x, p.Y + y, Color.Gray);
                  }
               }
            }
            */
            found = true;
         }
         if (!bigEnough(p, color))
         {
            /*
            badMarkers.ElementAt(num).Add(p);
            badMarkersExp.ElementAt(num).Add("Not big enough");
            for (int x = -2; x <= 2; x++)
            {
               for (int y = -2; y <= 2; y++)
               {
                  if (p.X + x >= 0 && p.X + x < bitmaps.ElementAt(num).Width && p.Y + y >= 0 && p.Y + y < bitmaps.ElementAt(num).Height)
                  {
                     Color c = bitmaps.ElementAt(num).GetPixel(p.X + x, p.Y + y);
                     if (c != Color.Red && c != Color.White && c != Color.Orange)
                        bitmaps.ElementAt(num).SetPixel(p.X + x, p.Y + y, Color.Gray);
                  }
               }
            }
            */
            found = true;
         }
         if (!found)
         {
            maximaNoDup.Add(p);

            Color mColor = new Color();
            if (color == GREEN)
               mColor = Color.Red;
            else if (color == RED)
               mColor = Color.White;
            else if (color == BLUE)
               mColor = Color.Orange;

            for (int x = -2; x <= 2; x++)
            {
               for (int y = -2; y <= 2; y++)
               {
                  if (p.X + x >= 0 && p.X + x < bitmaps.ElementAt(num).Width && p.Y + y >= 0 && p.Y + y < bitmaps.ElementAt(num).Height)
                     bitmaps.ElementAt(num).SetPixel(p.X + x, p.Y + y, mColor);
               }
            }

            markers.ElementAt(num).Add(p);
```

```
            count++;
        }
        pointCount++;
    }
    //Console.WriteLine("Count: " + count);
    return count;
}

public bool differentEnough(int x, int y, int color)
{
    bool diff = false;

    for (int n = -differenceRange; n <= differenceRange; n += differenceRange)
    {
        for (int m = -differenceRange; m <= differenceRange; m += differenceRange)
        {
            if (n != 0 && m != 0)
                continue;
            if (x + n >= 0 && x + n < bitmap.Width && y + m >= 0 && y + m < bitmap.Height)
                if (pixelRGB(bitmap.GetPixel(x, y), color) >= pixelRGB(bitmap.GetPixel(x + n, y + m), color) + colorDifferenceAmount)
                    diff = true;
        }
    }

    return diff;
}

public bool bigEnough(Point p, int color)
{
    bool enough = false;
    int x = p.X;
    int y = p.Y;

    for (int n = -minCellSize; n <= minCellSize; n += minCellSize)
    {
        for (int m = -minCellSize; m <= minCellSize; m += minCellSize)
        {
            if (n != 0 && m != 0)
                continue;
            if (x + n >= 0 && x + n < bitmap.Width && y + m >= 0 && y + m < bitmap.Height)
            {
                if (analysisMethod == CUTOFF)
                {
                    if (pixelRGB(bitmap.GetPixel(x + n, y + m), color) >= colorMinVal)
                        enough = true;
                }
                else if (analysisMethod == DIFFERENCE)
                {
                    if (pixelRGB(bitmap.GetPixel(x, y), color) >= pixelRGB(bitmap.GetPixel(x + n, y + m), color) + colorDifferenceAmount)
                        enough = true;
                }
            }
        }
    }

    return enough;
}

public bool isBadColor(Color c)
{
    if (c.R > whiteSens && c.G > whiteSens && c.B > whiteSens)
        return true;
    if (Math.Abs(c.R - c.G) < graySens && Math.Abs(c.G - c.B) < graySens)
        return true;
    return false;
}

public bool isBadColor(Color c, int color)
{
    if (isBadColor(c))
        return true;
    if (color == GREEN)
        return !(c.G > c.R && c.G > c.B);
    if (color == RED)
        return !(c.R > c.G && c.R > c.B);
    if (color == BLUE)
        return !(c.B > c.R && c.B > c.G);
    return false;
}

public Point findMaxima(Point p, int color)
{
    Point orig = p;
    Point lastPoint = p;
    Point newPoint = p;
    int best = pixelRGB(bitmap.GetPixel(p.X, p.Y), color);
newPt:
    for (int x = -pixelSearchRange; x <= pixelSearchRange; x++)
    {
        for (int y = -pixelSearchRange; y <= pixelSearchRange; y++)
        {
            p.X = lastPoint.X + x;
            p.Y = lastPoint.Y + y;
```

```
                if (p.X > 0 && p.X < bitmap.Width && p.Y > 0 && p.Y < bitmap.Height)
                {
                    if (pixelRGB(bitmap.GetPixel(p.X, p.Y), color) > best && !isBadColor(bitmap.GetPixel(p.X, p.Y)))
                    {
                        best = pixelRGB(bitmap.GetPixel(p.X, p.Y), color);
                        newPoint = p;
                    }
                }
            }
        }
        if (Math.Sqrt(Math.Pow(orig.X - lastPoint.X, 2) + Math.Pow(orig.Y - lastPoint.Y, 2)) > maxCellSize)
        {
            return lastPoint;
        }
        if (lastPoint != newPoint)
        {
            lastPoint = newPoint;
            goto newPt;
        }
        return lastPoint;
    }

    public int pixelRGB(Color c, int color)
    {
        if (color == GREEN)
            return c.G;
        if (color == RED)
            return c.R;
        if (color == BLUE)
            return c.B;
        return 0;
    }

    public void updateResults()
    {
        resultBox.Text = "";
        for (int i = 0; i < bitmaps.Count; i++)
        {
            resultBox.Text += "Pic " + (i + 1) + " of " + bitmaps.Count + ":" + Environment.NewLine;
            if (cellMult != 1)
            {
                if (lookingForGreen || results.ElementAt(i)[GREEN] > 0)
                {
                    resultBox.Text += " Green: " + results.ElementAt(i)[GREEN] + "  x" + cellMult + "  x" + volume + "  =  " + (int)(results.ElementAt(i)[GREEN] * cellMult *
volume) + Environment.NewLine;
                }
                if (lookingForRed || results.ElementAt(i)[RED] > 0)
                {
                    resultBox.Text += " Red:    " + results.ElementAt(i)[RED] + "  x" + cellMult + "  x" + volume + "  =  " + (int)(results.ElementAt(i)[RED] * cellMult * volume) +
Environment.NewLine;
                }
                if (lookingForBlue || results.ElementAt(i)[BLUE] > 0)
                {
                    resultBox.Text += " Blue:    " + results.ElementAt(i)[BLUE] + "  x" + cellMult + "  x" + volume + "  =  " + (int)(results.ElementAt(i)[BLUE] * cellMult * volume)
+ Environment.NewLine;
                }
            }
            else
            {
                if (lookingForGreen || results.ElementAt(i)[GREEN] > 0)
                {
                    resultBox.Text += " Green: " + results.ElementAt(i)[GREEN] + Environment.NewLine;
                }
                if (lookingForRed || results.ElementAt(i)[RED] > 0)
                {
                    resultBox.Text += " Red:    " + results.ElementAt(i)[RED] + Environment.NewLine;
                }
                if (lookingForBlue || results.ElementAt(i)[BLUE] > 0)
                {
                    resultBox.Text += " Blue:    " + results.ElementAt(i)[BLUE] + Environment.NewLine;
                }
            }
            foreach (Area a in areas)
            {
                resultBox.Text += " " + a.bitmapSiz.X + " x " + a.bitmapSiz.Y + "px";
            }
            resultBox.Text += Environment.NewLine;
        }
    }

    public void clearPics()
    {
        markers = new List<List<Point>>();
        badMarkers = new List<List<Point>>();
        badMarkersExp = new List<List<string>>();
        if (zoomLevel < 0)
            for (int i = zoomLevel; i < 0; i++)
                zoomInButton_Click(this, new EventArgs());
        if (zoomLevel > 0)
            for (int i = zoomLevel; i > 0; i--)
                zoomOutButton_Click(this, new EventArgs());
        pictureBox.Image = null;
        pictureBox.Size = new Size(602, 480);
```

```
    bitmap = null;
    foreach (Bitmap b in bitmaps)
        b.Dispose();
    bitmaps = new List<Bitmap>();
    origBitmaps = new List<Bitmap>();
    results = new List<int[]>();
    resultBox.Text = "";
    resultBox.Location = new Point(pictureBox.Location.X + pictureBox.Size.Width + 6, resultBox.Location.Y);
    picNames = new List<string>();
    nameBox.Text = "";
    currentPic = 0;
    prevButton.Enabled = false;
    nextButton.Enabled = false;
    savePicsButton.Enabled = false;

    clearAreaButton_Click(this, new EventArgs());
}

#region Form Stuff
public void Form1_MouseDown(object sender, MouseEventArgs e)
{
    mousePressed = true;
}

private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    mousePressed = false;
}

private void pictureBox_Click(object sender, EventArgs e)
{
    if (selectingArea || bitmaps.Count == 0 || activeArea != null)
        return;

    int[] r = results.ElementAt(currentPic);
    Color mColor = new Color();
    if (lookingForGreen)
    {
        r[GREEN]++;
        mColor = Color.Red;
    }
    else if (lookingForRed)
    {
        r[RED]++;
        mColor = Color.White;
    }
    else if (lookingForBlue)
    {
        r[BLUE]++;
        mColor = Color.Orange;
    }
    Point loc = this.PointToClient(Cursor.Position);
    loc.Offset(-pictureBox.Location.X, -pictureBox.Location.Y);
    if (zoomLevel < 0)
        for (int i = zoomLevel; i < 0; i++)
        {
            loc = new Point((int)(loc.X * 1.5), (int)(loc.Y * 1.5));
        }
    if (zoomLevel > 0)
        for (int i = zoomLevel; i > 0; i--)
        {
            loc = new Point((int)(loc.X * 2 / 3), (int)(loc.Y * 2 / 3));
        }
    for (int x = -2; x <= 2; x++)
    {
        for (int y = -2; y <= 2; y++)
        {
            if (loc.X + x >= 0 && loc.X + x < bitmaps.ElementAt(currentPic).Width && loc.Y + y >= 0 && loc.Y + y < bitmaps.ElementAt(currentPic).Height)
                bitmaps.ElementAt(currentPic).SetPixel(loc.X + x, loc.Y + y, mColor);
        }
    }

    markers.ElementAt(currentPic).Add(loc);
    results.Insert(currentPic, r);
    results.RemoveAt(currentPic + 1);
    updateResults();
}

private void progressPic_Click(object sender, EventArgs e)
{
    OpenFileDialog ofile = new OpenFileDialog();
    ofile.Filter = "Image File (*.bmp, *.jpg, *.png)|*.bmp;*.jpg;*.png";
    if (DialogResult.OK == ofile.ShowDialog())
    {
        progressPic.Image = new Bitmap(ofile.FileName);
    }
}

private void advancedPanel_MouseDown(object sender, MouseEventArgs e)
{
    movingSettings = true;
    settingsOffset = new Point(Cursor.Position.X - advancedPanel.Location.X, Cursor.Position.Y - advancedPanel.Location.Y);
}
```

```
private void advancedPanel_MouseUp(object sender, MouseEventArgs e)
{
    movingSettings = false;
}
#endregion

#region Main Buttons
private void selectPicButton_Click(object sender, EventArgs e)
{
    OpenFileDialog ofile = new OpenFileDialog();
    ofile.Multiselect = true;
    ofile.Filter = "Image File (*.bmp, *.jpg, *.png)|*.bmp;*.jpg;*.png";
    if (DialogResult.OK == ofile.ShowDialog())
    {
        clearPics();
        foreach (String file in ofile.FileNames)
        {
            bitmaps.Add(new Bitmap(file));
            origBitmaps.Add(new Bitmap(file));
            picNames.Add(file);
            markers.Add(new List<Point>());
            badMarkers.Add(new List<Point>());
            badMarkersExp.Add(new List<string>());
            results.Add(new int[]{0, 0, 0});
        }
        pictureBox.Image = bitmaps.ElementAt(0);
        pictureBox.Size = new System.Drawing.Size(pictureBox.Image.Width, pictureBox.Image.Height);

        nameBox.Text = "1 of " + bitmaps.Count + ": " + picNames.ElementAt(0);
        bitmap = bitmaps.ElementAt(0);
        resultBox.Location = new Point(pictureBox.Location.X + pictureBox.Size.Width + 6, resultBox.Location.Y);
    }
    if (bitmaps.Count > 1)
    {
        prevButton.Enabled = true;
        nextButton.Enabled = true;
    }
    else
    {
        prevButton.Enabled = false;
        nextButton.Enabled = false;
    }
    savePicsButton.Enabled = false;
}

private void analyzeButton_Click(object sender, EventArgs e)
{
    if (bitmaps.Count == 0)
        return;
    bitmaps = new List<Bitmap>();
    foreach (Bitmap b in origBitmaps)
        bitmaps.Add(new Bitmap(b));
    analyze();
    savePicsButton.Enabled = true;
}

private void clearButton_Click(object sender, EventArgs e)
{
    clearPics();
}

private void prevButton_Click(object sender, EventArgs e)
{
    int thisZoom = zoomLevel;
    if (bitmaps.Count == 0)
        return;
    //last pic regular zoom
    if (zoomLevel < 0)
        for (int i = zoomLevel; i < 0; i++)
            zoomInButton_Click(this, new EventArgs());
    if (zoomLevel > 0)
        for (int i = zoomLevel; i > 0; i--)
            zoomOutButton_Click(this, new EventArgs());
    if (currentPic == 0)
        currentPic = bitmaps.Count - 1;
    else
        currentPic--;
    //new pic preferred zoom
    pictureBox.Image = bitmaps.ElementAt(currentPic);
    pictureBox.Size = new System.Drawing.Size(pictureBox.Image.Width, pictureBox.Image.Height);

    zoomLevel = 0;
    if (thisZoom < 0)
        for (int i = thisZoom; i < 0; i++)
            zoomOutButton_Click(this, new EventArgs());
    if (thisZoom > 0)
        for (int i = thisZoom; i > 0; i--)
            zoomInButton_Click(this, new EventArgs());
    nameBox.Text = (currentPic + 1) + " of " + bitmaps.Count + ": " + picNames.ElementAt(currentPic);
    resultBox.Location = new Point(pictureBox.Location.X + pictureBox.Size.Width + 6, resultBox.Location.Y);
}
```

```
private void nextButton_Click(object sender, EventArgs e)
{
    int thisZoom = zoomLevel;
    if (bitmaps.Count == 0)
        return;
    //last pic regular zoom
    if (zoomLevel < 0)
        for (int i = zoomLevel; i < 0; i++)
            zoomInButton_Click(this, new EventArgs());
    if (zoomLevel > 0)
        for (int i = zoomLevel; i > 0; i--)
            zoomOutButton_Click(this, new EventArgs());
    if (currentPic == bitmaps.Count - 1)
        currentPic = 0;
    else
        currentPic++;
    //new pic preferred zoom
    pictureBox.Image = bitmaps.ElementAt(currentPic);
    pictureBox.Size = new System.Drawing.Size(pictureBox.Image.Width, pictureBox.Image.Height);

    zoomLevel = 0;
    pictureBox.Refresh();
    pictureBox.Show();
    if (thisZoom < 0)
        for (int i = thisZoom; i < 0; i++)
            zoomOutButton_Click(this, new EventArgs());
    if (thisZoom > 0)
        for (int i = thisZoom; i > 0; i--)
            zoomInButton_Click(this, new EventArgs());
    nameBox.Text = (currentPic + 1) + " of " + bitmaps.Count + ": " + picNames.ElementAt(currentPic);
    nameBox.Refresh();
    nameBox.Show();
    resultBox.Location = new Point(pictureBox.Location.X + pictureBox.Size.Width + 6, resultBox.Location.Y);
}

private void zoomInButton_Click(object sender, EventArgs e)
{
    if (analyzing || bitmaps.Count == 0)
        return;
    pictureBox.Size = new Size((int)(pictureBox.Size.Width * 1.5), (int)(pictureBox.Size.Height * 1.5));
    resultBox.Location = new Point(pictureBox.Location.X + pictureBox.Size.Width + 6, resultBox.Location.Y);
    foreach (Area a in areas)
    {
        Point p = new Point(a.location.X - pictureBox.Location.X, a.location.Y - pictureBox.Location.Y);
        p = new Point((int)(p.X * 1.5), (int)(p.Y * 1.5));
        p = new Point(p.X + pictureBox.Location.X, p.Y + pictureBox.Location.Y);
        a.zoomArea(p, new Point((int)(a.size.X * 1.5), (int)(a.size.Y * 1.5)));
    }
    zoomLevel++;
}

private void zoomOutButton_Click(object sender, EventArgs e)
{
    if (analyzing || bitmaps.Count == 0)
        return;
    pictureBox.Size = new Size((int)(pictureBox.Size.Width * 2 / 3), (int)(pictureBox.Size.Height * 2 / 3));
    resultBox.Location = new Point(pictureBox.Location.X + pictureBox.Size.Width + 6, resultBox.Location.Y);
    foreach (Area a in areas)
    {
        Point p = new Point(a.location.X - pictureBox.Location.X, a.location.Y - pictureBox.Location.Y);
        p = new Point((int)(p.X * 2 / 3), (int)(p.Y * 2 / 3));
        p = new Point(p.X + pictureBox.Location.X, p.Y + pictureBox.Location.Y);
        a.zoomArea(p, new Point((int)(a.size.X * 2 / 3), (int)(a.size.Y * 2 / 3)));
    }
    zoomLevel--;
}

private void advancedButton_Click(object sender, EventArgs e)
{
    advancedPanel.Visible = true;
    advancedPanel.BringToFront();
    advancedInfoBox.Text = "";
    greenCheckBox.Checked = lookingForGreen;
    redCheckBox.Checked = lookingForRed;
    blueCheckBox.Checked = lookingForBlue;

    greenValEntry.Text = colorMinVal + "";
    minCellDistEntry.Text = minCellDist + "";
    pixelSearchDistEntry.Text = pixelSearchRange + "";
    whiteSensEntry.Text = whiteSens + "";
    graySensEntry.Text = graySens + "";
    maxCellDistEntry.Text = maxCellDist + "";
    colorDiffRangeBox.Text = differenceRange + "";
    greenDiffAmountBox.Text = colorDifferenceAmount + "";
    minCellSizeBox.Text = minCellSize + "";
    edgeBufferBox.Text = edgeBuffer + "";
    defaultAreaSizeBox.Text = defaultAreaSize + "";
    cellMultiplierBox.Text = cellMult + "";
    totalVolBox.Text = volume + "";
    maxCellSizeBox.Text = maxCellSize + "";
    pixelScanEntry.Text = pixelScanVal + "";
}
```

```
        private void areaButton_Click(object sender, EventArgs e)
        {
            if (analyzing || bitmaps.Count == 0)
                return;
            selectingArea = true;
        }

        private void clearAreaButton_Click(object sender, EventArgs e)
        {
            foreach (Area a in areas)
                a.dispose();
            areas = new List<Area>();
        }

        private void defaultAreaButton_Click(object sender, EventArgs e)
        {
            if (analyzing || bitmaps.Count == 0)
                return;
            defaultArea = true;
            //activeArea = new Area(new Point(this.PointToClient(Cursor.Position).X, this.PointToClient(Cursor.Position).Y), new Point(defaultAreaSize, defaultAreaSize));
            activeArea = new Area(new Point(this.PointToClient(Cursor.Position).X - (int)(defaultAreaSize / 1), this.PointToClient(Cursor.Position).Y - (int)(defaultAreaSize / 1)),
new Point((int)(defaultAreaSize * Math.Pow(1.5, zoomLevel)), (int)(defaultAreaSize * Math.Pow(1.5, zoomLevel))));
        }

        private void mainHelpButton_Click(object sender, EventArgs e)
        {
            helpPanel.Visible = !helpPanel.Visible;
            helpPanel.BringToFront();
        }

        private void savePicsButton_Click(object sender, EventArgs e)
        {
            if (results.Count != bitmaps.Count)
            {
                MessageBox.Show("Please run the analysis to save pictures");
                return;
            }

            string path = "";
            FolderBrowserDialog fbd = new FolderBrowserDialog();
            fbd.SelectedPath = picNames.ElementAt(0).Substring(0, picNames.ElementAt(0).LastIndexOf("\\"));
            DialogResult result = fbd.ShowDialog();
            if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(fbd.SelectedPath))
            {
                path = fbd.SelectedPath;

                for (int i = 0; i < bitmaps.Count; i++)
                {
                    int start = picNames.ElementAt(i).LastIndexOf("\\");
                    int end = picNames.ElementAt(i).Length - 4;
                    bitmaps.ElementAt(i).Save(path + picNames.ElementAt(i).Substring(start, end - start) + "-" + results.ElementAt(i).Sum() + ".png",
System.Drawing.Imaging.ImageFormat.Png);
                    workingAnimation();
                }
                if (bitmaps.Count == 1)
                    MessageBox.Show("Picture successfully saved");
                else
                    MessageBox.Show("Pictures successfully saved");
            }
        }
        #endregion

        #region Settings Buttons
        private void advancedAcceptButton_Click(object sender, EventArgs e)
        {
            advancedPanel.Visible = false;
            lookingForGreen = greenCheckBox.Checked;
            lookingForRed = redCheckBox.Checked;
            lookingForBlue = blueCheckBox.Checked;
            saveValues();
        }

        private void advancedCancelButton_Click(object sender, EventArgs e)
        {
            advancedPanel.Visible = false;
        }

        private void greenValEntry_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                advancedAcceptButton_Click(this, new EventArgs());
            }
        }

        private void methodBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            analysisMethod = methodBox.SelectedIndex;
            if (analysisMethod == CUTOFF)
            {
                colorDiffRangeBox.ReadOnly = true;
                greenDiffAmountBox.ReadOnly = true;
                greenValEntry.ReadOnly = false;
```

```
        }
        else if (analysisMethod == DIFFERENCE)
        {
            colorDiffRangeBox.ReadOnly = false;
            greenDiffAmountBox.ReadOnly = false;
            greenValEntry.ReadOnly = true;
        }
    }

    private void methodInfoButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "The Cutoff method detects cells based on the amount of color they have.  The Difference method detects cells based on the difference in color between them and their surroundings.";
    }

    private void greenValHelp_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Only 0-255) If your background is closer to the cell color, set this higher so the program doesn't see the background as a cell.  If your cells are a dimmer color, set this lower to detect them.";
    }

    private void graySensButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Only 0-255) This keeps gray colors from being recognized as cells.  If your cells are grayer in color, this may need to be lowered.";
    }

    private void whiteSensButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Only 0-255) This keeps white colors from being recognized as cells. Raise this number for whiter cells, lower it to keep from counting non-cell white spots.";
    }

    private void pixelScanButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 1 - 4) This is how closely the program searches the pixels.  Higher numbers greatly speed up the program.  Lower numbers needed for very small cells or low resolution images.";
    }

    private void minCellDistButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 5-20) This keeps the program from seeing one cell as multiple.  If your cells are close together, make this small; if your cells are farther apart, make this larger";
    }

    private void maxCellDistButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 15-60) This lets the program run more quickly.  Smaller = Faster.  Should be at least 3X the Min Cell Distance.";
    }

    private void minCellSizeButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 2-10) This keeps the program from seeing small specks as cells.  If your cells are bigger or small specks are being detected, make this bigger.  If your cells are smaller, make this smaller.";
    }

    private void pixelSearchDistButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 1-3) If your picture is more grainy, make this bigger, but usually 1 works fine.  Bigger numbers make the program slower.";
    }

    private void colorDiffRangeButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 5-15) If your cells are closer together or have fine edges, make this smaller.  If your cells are more spread out or have fuzzy edges, make this larger.";
    }

    private void greenDiffAmountButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Only 0-255) If your cells are more similar in color to the background, make this smaller.";
    }

    private void edgeBufferButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "(Suggested 0-5) This gets rid of small blips on the edge or picture borders being seen as cells.  If there are incorrect edge counts, make this larger. If cells on the edge aren't being detected, make this smaller.";
    }

    private void defaultAreaSizeButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "This is the size of the default area that can be placed, in pixels.";
    }

    private void cellMultiplierButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "The number to multiply the number of cells by for recording data.";
    }

    private void totalVolButton_Click(object sender, EventArgs e)
    {
        advancedInfoBox.Text = "This number is used to multiply the number of cells for recording data.";
    }

    private void maxCellSizeButton_Click(object sender, EventArgs e)
```

```
    {
        advancedInfoBox.Text = "(Suggested 25-50) This keeps the program from seeing masses of cells as 1 cell.  If your cells are bigger, make this bigger.  If your cells are
smaller, make this smaller.";
    }

    private void saveSettingsButton_Click(object sender, EventArgs e)
    {
        SaveFileDialog sfile = new SaveFileDialog();
        sfile.Filter = "Text File (*.txt)|*.txt";
        if (DialogResult.OK == sfile.ShowDialog() && sfile.FileName != "")
        {
            saveValues();
            using (StreamWriter writer = new StreamWriter(sfile.OpenFile()))
            {
                writer.Write(colorMinVal + "," + graySens + "," + whiteSens + "," + minCellDist + "," + maxCellDist + "," +
                    minCellSize + "," + maxCellSize + "," + pixelSearchRange + "," + differenceRange + "," + colorDifferenceAmount
                    + "," + edgeBuffer  + "," + defaultAreaSize + "," + cellMult + "," + volume + "," + pixelScanVal);
                writer.Dispose();
                writer.Close();
            }
            int index = sfile.FileName.LastIndexOf("\\");
            savedSettingsLabel.Text = sfile.FileName.Substring(index + 1);
        }
    }

    private void loadSettingsButton_Click(object sender, EventArgs e)
    {
        OpenFileDialog ofile = new OpenFileDialog();
        ofile.Multiselect = false;
        ofile.Filter = "Text File (*.txt)|*.txt";
        if (DialogResult.OK == ofile.ShowDialog())
        {
            String str = System.IO.File.ReadAllText(ofile.FileName);

            int index = str.IndexOf(",");
            colorMinVal = Int32.Parse(str.Substring(0, index));
            greenValEntry.Text = colorMinVal + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            graySens = Int32.Parse(str.Substring(0, index));
            graySensEntry.Text = graySens + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            whiteSens = Int32.Parse(str.Substring(0, index));
            whiteSensEntry.Text = whiteSens + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            minCellDist = Int32.Parse(str.Substring(0, index));
            minCellDistEntry.Text = minCellDist + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            maxCellDist = Int32.Parse(str.Substring(0, index));
            maxCellDistEntry.Text = maxCellDist + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            minCellSize = Int32.Parse(str.Substring(0, index));
            minCellSizeBox.Text = minCellSize + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            maxCellSize = Int32.Parse(str.Substring(0, index));
            maxCellSizeBox.Text = maxCellSize + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            pixelSearchRange = Int32.Parse(str.Substring(0, index));
            pixelSearchDistEntry.Text = pixelSearchRange + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            differenceRange = Int32.Parse(str.Substring(0, index));
            colorDiffRangeBox.Text = differenceRange + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            colorDifferenceAmount = Int32.Parse(str.Substring(0, index));
            greenDiffAmountBox.Text = colorDifferenceAmount + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            edgeBuffer = Int32.Parse(str.Substring(0, index));
            edgeBufferBox.Text = edgeBuffer + "";
            str = str.Substring(index + 1);

            index = str.IndexOf(",");
            defaultAreaSize = Int32.Parse(str.Substring(0, index));
            defaultAreaSizeBox.Text = defaultAreaSize + "";
            str = str.Substring(index + 1);
```

```
            index = str.IndexOf(",");
            cellMult = Int32.Parse(str.Substring(0, index));
            cellMultiplierBox.Text = cellMult + "";
            str = str.Substring(index + 1);

            if (str.IndexOf(",") == -1)
            {
                volume = Double.Parse(str);
                totalVolBox.Text = volume + "";
                pixelScanVal = 2;
                goto done;
            }

            index = str.IndexOf(",");
            volume = Double.Parse(str.Substring(0, index));
            totalVolBox.Text = volume + "";
            str = str.Substring(index + 1);

            if (str.IndexOf(",") == -1)
            {
                pixelScanVal = Int32.Parse(str);
                pixelScanEntry.Text = pixelScanVal + "";
                goto done;
            }

            index = str.IndexOf(",");
            pixelScanVal = Int32.Parse(str.Substring(0, index));
            pixelScanEntry.Text = pixelScanVal + "";
            str = str.Substring(index + 1);

            //Add additional setting vals here

        done:
            index = ofile.FileName.LastIndexOf("\\");
            savedSettingsLabel.Text = ofile.FileName.Substring(index + 1);
        }
    }

    private void saveValues()
    {
        colorMinVal = Convert.ToInt32(greenValEntry.Text);
        minCellDist = Convert.ToDouble(minCellDistEntry.Text);
        pixelSearchRange = Convert.ToInt32(pixelSearchDistEntry.Text);
        whiteSens = Convert.ToInt32(whiteSensEntry.Text);
        graySens = Convert.ToInt32(graySensEntry.Text);
        maxCellDist = Convert.ToDouble(maxCellDistEntry.Text);
        differenceRange = Convert.ToInt32(colorDiffRangeBox.Text);
        colorDifferenceAmount = Convert.ToInt32(greenDiffAmountBox.Text);
        minCellSize = Convert.ToInt32(minCellSizeBox.Text);
        edgeBuffer = Convert.ToInt32(edgeBufferBox.Text);
        defaultAreaSize = Convert.ToInt32(defaultAreaSizeBox.Text);
        cellMult = Convert.ToDouble(cellMultiplierBox.Text);
        volume = Convert.ToDouble(totalVolBox.Text);
        maxCellSize = Convert.ToInt32(maxCellSizeBox.Text);
        pixelScanVal = Convert.ToInt32(pixelScanEntry.Text);
    }
    #endregion

}

class Area
{
    public Point location;
    public Point size;
    public Point bitmapLoc;
    public Point bitmapSiz;
    public Label top;
    public Label bottom;
    public Label left;
    public Label right;

    public Area()
    {

    }

    public Area(Point loc, Point siz)
    {
        location = loc;
        size = siz;

        top = new Label();
        bottom = new Label();
        left = new Label();
        right = new Label();

        List<Label> list = new List<Label>{top, bottom, left, right};
        foreach (Label l in list)
        {
            l.BackColor = Color.Orange;
            l.Text = "";
            l.AutoSize = false;
```

```
            l.Parent = Form1.ActiveForm;
            l.BorderStyle = BorderStyle.None;
            l.Visible = true;
            l.MouseDown += new MouseEventHandler(Form1.me.Form1_MouseDown);
        }

        redoLocSize(true);
    }

    public Area(Point upperLeft, Point lowerRight, int nothing) :
        this(upperLeft, new Point(lowerRight.X - upperLeft.X, lowerRight.Y - upperLeft.Y))
    {

    }

    public void moveArea(Point loc, Point siz)
    {
        location = loc;
        size = siz;

        redoLocSize(true);
    }

    public void moveArea(Point upperLeft, Point lowerRight, int nothing)
    {
        moveArea(upperLeft, new Point(lowerRight.X - upperLeft.X, lowerRight.Y - upperLeft.Y));
    }

    public void zoomArea(Point loc, Point siz)
    {
        location = loc;
        size = siz;

        redoLocSize(false);
    }

    private void redoLocSize(bool doIt)
    {
        if (doIt)
        {
            bitmapLoc = new Point(location.X - Form1.me.pictureBox.Location.X, location.Y - Form1.me.pictureBox.Location.Y);
            bitmapSiz = size;
        }

        if (size.X >= 0)
        {
            if (size.Y >= 0)
            {
                top.Location = new Point(location.X - 6, location.Y - 6);
                left.Location = new Point(location.X - 6, location.Y - 6);
                bottom.Location = new Point(location.X - 6, location.Y + size.Y);
                right.Location = new Point(location.X + size.X, location.Y - 6);
            }
            else
            {
                top.Location = new Point(location.X - 6, location.Y + size.Y - 6);
                left.Location = new Point(location.X - 6, location.Y + size.Y - 6);
                bottom.Location = new Point(location.X - 6, location.Y);
                right.Location = new Point(location.X + size.X, location.Y + size.Y - 6);

                location = new Point(location.X, location.Y + size.Y);
            }
        }
        else
        {
            if (size.Y >= 0)
            {
                top.Location = new Point(location.X + size.X - 6, location.Y - 6);
                left.Location = new Point(location.X + size.X - 6, location.Y - 6);
                bottom.Location = new Point(location.X + size.X - 6, location.Y + size.Y);
                right.Location = new Point(location.X, location.Y - 6);

                location = new Point(location.X + size.X, location.Y);
            }
            else
            {
                top.Location = new Point(location.X + size.X - 6, location.Y + size.Y - 6);
                left.Location = new Point(location.X + size.X - 6, location.Y + size.Y - 6);
                bottom.Location = new Point(location.X + size.X - 6, location.Y);
                right.Location = new Point(location.X, location.Y + size.Y - 6);

                location = new Point(location.X + size.X, location.Y + size.Y);
            }
        }

        size = new Point(Math.Abs(size.X), Math.Abs(size.Y));

        top.Size = new Size(Math.Abs(size.X) + 12, 6);
        bottom.Size = new Size(Math.Abs(size.X) + 12, 6);
        right.Size = new Size(6, Math.Abs(size.Y) + 12);
        left.Size = new Size(6, Math.Abs(size.Y) + 12);

        top.BringToFront();
```

```
            bottom.BringToFront();
            left.BringToFront();
            right.BringToFront();
        }

        public void dispose()
        {
            top.Dispose();
            bottom.Dispose();
            left.Dispose();
            right.Dispose();
        }
    }
}
```