TANAY KANE SENIOR FOCUS PORTFOLIO

May 21, 2019

Tanay Kane

Contents

1. First Semester

1.1. Statement of Courses Dropped:

Literature & Film:

(1) D Block, Monday

Bio Ethics:

(1) H Block, Tuesday

1.2. First Semester Guiding Questions

(1) What is computer vision?
    (a) What is image classification
(2) What are the applications for computer vision?
(3) Why are neural networks used for computer vision?
(4) What are some of the neural networks used for computer vision?
    (a) Convolutional Neural networks (CNN)
    (b) CNN-RNN
(5) How does each neural network work?
    (a) CNN
        (i) What are the layers of a Convolutional Nerual network?
    (b) CNN-RNN
        (i) What are recurrent networks?
            (A) What are LSTMs?
(6) How does each network process an image?
(7) What are the advantages of each network?
(8) What are the disadvantages of each network?
(9) How do you assess a model for image classification?

1.3. First Semester Paper

The Optimal Neural Network for Computer Vision

Quantitative Paper

Glossary Items in **Bold**

Tanay Kane

I. Introduction

On December 15, 2016, Amazon surprised the tech world when they announced Amazon Go, a new take on grocery shopping. Instead of the traditional system where a customer would have to go to a cashier and pay for their items, Amazon Go promised a truly "cashier- less" system where customers would just open an app, shop for whatever they wanted, and then be billed once they left [30]. How did they keep track of what people bought? According to Amazon, they utilized strategically placed cameras that would continuously take a video feed of the store. The data from these cameras would then be fed into a computer vision algorithm, which analyzed what each customer bought. These items were then added to a list and the list would keep updating as the customer continued to shop [30]. When he or she left, that algorithm would calculate the customer's bill and send it to them via the app. This "revolutionary" system is just one of many examples as to how computer vision is becoming a key part of the modern world.

Computer vision is the study of how a computer can be programmed to obtain information from images. In recent years, computer vision has risen in popularity as many companies such as Tesla, Instagram and Facebook have been utilizing it for various tasks [16] [27]. One of the most popular tasks is image classification. In image classification, a computer vision algorithm analyzes an image in order to identify what is it contains. Image classification has been used by many big companies for a variety of reasons. For example, for Tesla, image classification has been crucial for piloting their self-driving cars by identifying pedestrians and traffic signs, and for Instagram and Facebook, image classification has become a key part in facial recognition as well as filtering inappropriate pictures. In order to perform these tasks, a popular approach among computer vision researchers is to utilize deep learning algorithms.

Deep learning is a subset of machine learning which deals with the use of multi-layer structures to process information [21]. Because of their multi-layer structures, deep learning algorithms require more data since they are more complex than other machine learning algorithms. Most modern deep learning algorithms are based around an artificial neural network, a network that consists of multiple connected layers that are made up of individual "neurons" [9]. Each "neuron" performs a certain mathematical operation using its input, a certain amount of **weight and a bias factor**. The mathematical operation varies depending on the type of problem that the network is being applied to or type of network that is being built [9]. These structures are very complex as multiple layers of these neruons are intertwined in many different ways, and as such, each neuron requrires a lot of data and more processing power. However, although

deep learning algoithms consume a lot of memory, they are continually used in many computer vision applications. These algorithms have popular as they been proven to obtain more accurate results in a shorter time span when used with **GPU processing** [12].

Within the field of computer vision, there is a major debate as to what the "optimal" deep learning algorithm for image classification is. The "optimal" algorithm would have to successfully identify parts of an image, while also being a practical algorithm for real world applications. Practicality is measured by how much memory is consumed by the algorithm, and how much time it takes to train the algorithm. With this metric, a network not only has to score a lower error rate, but also be effective in how it allocates memory and how fast it runs. However, since the hardware that each respective model is being run on can affect these values, the efficiency in time and memory has to be taken as a general case. Even though there is a plethora of algorithms, each with their own respective architectures, two main networks stand out: convolutional neural network, and the CNN-RNN. The purpose of this paper is to explore both algorithms and identify which one is the optimal algorithm.

II. Convolutional Neural Networks

Convolutional neural networks are the most popular algorithm when it comes to computer vision. They are a special type of artificial neural networks that were made with the intention of image processing [19]. The convolutional neural network was first introduced by Kunihiko Fukushima in his paper *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position* [25]. In this paper, Fukushima proposed a network that was based on Hubel and Wiesel's hierarchical structure of the nervous system that was responsible for human vision [7] and called his network the "Neocognitron" (as it was an extension of the "Cognitron" model that he had proposed earlier in his career) [7]. The "Neocognitron" was made up of a photoreceptor array, and layers that consisted of S-cell and C-cell neurons. The photoreceptor array would take in as input an array of numbers, each ranging from 0-255, that would correspond to the pixel values of the image. After taking the image as input, the layers made up of "S cells" and "C cells" would then act like filters on the array [7]. "S-cells" would capture any distinguishing features of the image, while the C-cell would try and identify deformities within the features that the S-cell had identified. These features would then culminate into a set of confidence values, one for each possible label that the image could be classified as, where the largest value corresponds to what the algorithm identified the image as.

The Neocognitron was so successful at classifying images, that it has become the basis for all modern convolutional neural networks [25]. However, as time passed on the S-cells and C-cells of the Neocognitron evolved into different layers that could each perform individual tasks. In order to assess whether the convolutional neural networks is the"optimal" network, it is important to understand each of the layers of the convolutional neural network and what they contribute. To understand these concepts, it helps to not only go through the layer, but walk through a full example of the network in action.

A. Layers

In order to understand modern Convolutional Neural networks, it is important to see the different layers of the network and how they interact. The easiest way to understand these layers and how they interact is to work through one of the most common convolutional network architecture: $INPUT \rightarrow$

$[[CONV \rightarrow RELU]*N \rightarrow POOL]*M \rightarrow [FC \rightarrow RELU] * K \rightarrow FC$ [19]. In this representation, INPUT refers to the input layer, CONV refers to the convolutional layer, RELU refers to the rectifier layer, POOL refers to the pooling layer and FC refers to the fully connected layer. The letters N, M, and K refer to an arbitrary value that would represent how many times each combination would be repeated [19]. Although different architectures exist, each with their own specialized layer, most, if not all, modern convolutional neural networks use these layers.

1) Convolutional Layer: The convolutional layer is based on the mathematical concept of a convolution. A convolution is an operation that takes two functions as inputs and returns a function that best represents both of the input functions [9]. In turn, the convolutional layer performs a similar action. The convolutional layer takes in the input matrix as its first function and an arbitrary filter as its second function. The layer then returns a matrix whose components are simply the dot product of the filter and a certain section of the input matrix [26]. In this case, the dot product acts as the best representation for how the image reacts to the filter function. The convolutional layer is beneficial for three main reasons: *equivariance*, *sparse interaction*, and *parameter sharing*.

Equivariance is a mathematical term and refers to the sensitivity of a function. A function is equivariant if when the input is augmented in a certain way, the output of that function is augmented in a similar way [9]. An equivariant function proves to be very advantageous as the function is able to mimic any changes that may have been done to an image. For example, if one fed a convolutional neural network an image of a cat and then fed the same network an image of the same cat but shifted to the right, the convolutional layer's equivariance would allow it to classify both images as a cat. This quality is very helpful for computer vision applications as it reduces the chances of an augmented image being misclassified [9].

In the convolutional layer, and with convolutional neural networks in general, each neuron that makes up the output layer is always less than or equal to the previous layer [9] [19] . This feature is known as *sparse interaction* because the fewer neurons there are, the less individual interactions occur. Figure 1 illustrates this feature as the number of h neurons are less than the number of x neurons. Sparse interaction is beneficial as it saves the network a lot of time and memory. Since each neuron contains little bits of information that has to be stored in memory, decreasing the number of neurons at each step is a big advantage as it saves the network a lot of memory, as less information has to be stored. Sparse interaction is also beneficial as it decreases the amount of time it takes for the network to run. Because there are less neurons in each successive layer, the number of total computations is decreased [19] .

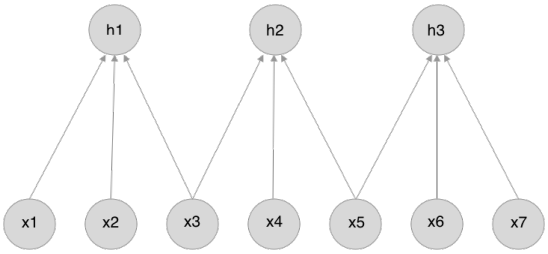

Fig. 1: A drawing of how a convolutional layer would interact with it's output layer. In this diagram, x 1 through 7 represent the input neurons and h 1 through 3 represent output neurons.

Parameter sharing, more commonly known as local connectivity, refers to the connections of each of the neuron in the layer. Contrary to sparse interaction, parameter sharing refers to the smaller number of connections rather than the number of neurons [9]. In the convolution layer, each input neuron is

only connected to the N closest output neurons. In this case, N refers to how many connections each input neuron should make [19]. For example, in Figure 1, the neuron x3 is only connected to h1 and h2, rather than it being connected to every h neuron. These few connections allow the convolutional layer to save a lot of time as well as memory [21]. Since each of the connections require a bit of memory to compute, a layer that is filled with connections would require more memory as more values are needed to continually perform each individual operation [15]. However, parameter sharing allows the convolutional layer to use up less memory by allowing the network to store less data. Since parameter sharing saves on memory, the feature also saves time as the more memory a process needs, the more time it needs to complete said process [21].

2) Rectifier Layer: The rectifier layer is a layer that computes the Rectified Linear Unit, ReLu for short, activation function. Activation functions compute a value that corresponds to whether or not the values in a neuron should be used [19]. There are many different activations that exist, but the ReLu activation function is the most effective for the convolutional neural network. [21]. The ReLu function takes in a matrix and a filter as input and outputs a matrix where each component is the greatest value of the filter matrix at a point ,unless there is a negative value. If there is a negative value, ReLu will return 0.

$$\begin{bmatrix} 1 & 3 & 4 & 5 \\ -6 & 0 & 8 & 2 \\ 5 & 0 & 9 & 4 \\ 7 & 3 & 1 & 8 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 8 & 8 \\ 0 & 9 & 9 \\ 7 & 9 & 9 \end{bmatrix}$$

In order to understand the ReLu function, it is best to work through an example. Let M be the 4×4 matrix in the equation above, represent the input value and let R be a 2×2 matrix, R, have a stride value 1. As R glides across M, moving according to the stride value, R will return the largest value of the 4 numbers it sees. R will continue to do this until it outputs the 3×3 matrix in the equation above. Because of the negative 6 in M, R has two components that are 0. These values then represent whether or not a certain neuron is "activated" or not. In the output matrix, each value represents the activation value of neuron. Since there are two zeroes, two out of the nine total neurons are not activated and therefore ignored. The other seven neurons would then correspond to certain features of the image. For example, on the picture of a handwritten five, the ReLu layer would identify the defining characteristics of the five, such as the the line at the top or the curve at the bottom.

The main advantage of the ReLu function is that it maximizes speed of the network by reducing the amount of time it takes to compute the **gradient**. Gradients are a key part in assessing the effectiveness of the layers of a neural network. The gradient is obtained after **backpropagation** is done to the layer. During backpropagation, the network assess whether or not the weights and biases of a neuron is correct by computing the gradient of the neurons [21]. With more complex activation functions, the gradient becomes just as complex and therefore there is more room for errors [9]. However, because of the way the ReLu function behaves, its gradient can only be between 0 and 1. This simplicity allows the network to calculate the ReLu's function faster, which greatly decreases the time it takes for the network to run.

3) Pooling Layer: The pooling layer's function is to decrease the input matrix to only represent the "important" parts. In convolutional neural networks, the pooling layer implements the max function in order to shrink the input matrix [19]. Max pooling is very similar to the ReLu function, the only difference being how each function treats negative

values. Whereas the ReLu function returns 0 if there is a negative number, the max pooling layer takes negative numbers into account [24].

$$\begin{bmatrix} 1 & 3 & 4 & 5 \\ -6 & 0 & 8 & 2 \\ 5 & 0 & -9 & -4 \\ 7 & 3 & -1 & -8 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 & 8 & 8 \\ 5 & 8 & 8 \\ 7 & 3 & -1 \end{bmatrix}$$

The matrices above show the max pooling layer in action. In the equation above, the pooling layer is fed the matrix M, the 4×4 matrix, as well as a matrix, R, a 2×2 matrix with a stride value of 1. As R glides across M, R looks at a section of M and returns the largest value. When R has gone through M, the 3×3 matrix above is produced. In a numerical sense, the matrix that was produced in the equation above only contains the most important parts of figure M, as the values that are considered important are just the largest values, which correspond to certain features of an image. For example, if an image of a man with a tree in the background were to be fed into the model and the goal was to identify a face, the pooling layer would shrink the image to focus on the man's face rather than the trees.

The pooling layer optimizes the convolutional neural network by using less memory [24]. This shrinking provides two main advantages for the convolutional neural network: it saves memory and time, and it cuts out noise for future processing. By shrinking the input matrix, the amount of calculations that have to be done by the network also decreases as the matrix itself is smaller. The pooling layer also makes sure that shrinking the matrix does not ignore any important information that may be passed into it [9]. Since the number of calculations decrease, the network's training time is drastically smaller [3]. With this reduced training time, the neural network becomes more efficient as it can process bigger chunks of data faster. More often than not, these big chunks of data have a lot of unnecessary details. The pooling layer's shrinking cuts out these two by only focusing on features that are important to the task at hand.

4) Fully Connected Layer: The fully connected layer is the only layer that is not unique to the convolutional neural network and is always the last layer of the network. In this layer, each neuron is connected to every neuron of its input and uses those values to compute a special activation function [19]. This idea is best illustrated in Figure 2 as each of the h neurons are connected to each of the x neurons. For each h neuron in Figure 2, there is a set of weights and biases that are associated with each connection and an activation function that has to be computed [19]. First, the neuron takes in the inputs as a vector where each component is equal to the input neuron's value. The neuron then computes the dot product of the input matrix and the weights and biases. This value is then put into an activation function and the neuron returns the result.

In order to fully explain this layer, it is best to work through an example using Figure 2. Let the x layer be the output of the pooling layer that preceded it and each of the values be 1, 0.5, 0.75, and 0. The set of weights associated with each neuron, in order, are 0.2, 0.3, 0.7 and 0.6, and the h neurons will perform the **sigmoid activation function**:

$$S(x) = \frac{1}{1 + e^{-x}}$$

H1 will first compute the dot product between the x layer outputs and the vector of the weights which comes out to be: $1 \cdot 0.2 + 0.5 \cdot 0.3 + 0.75 \cdot 0.7 + 0 \cdot 0.6 = 2.225$. After we compute the dot product, we then input the result into the sigmoid function S(x): $S(2.225) = \frac{1}{1+e^{-2.225}} \approx 0.9025$ In a real world situation, this value would represent high level features of the image [9]. Higher level features

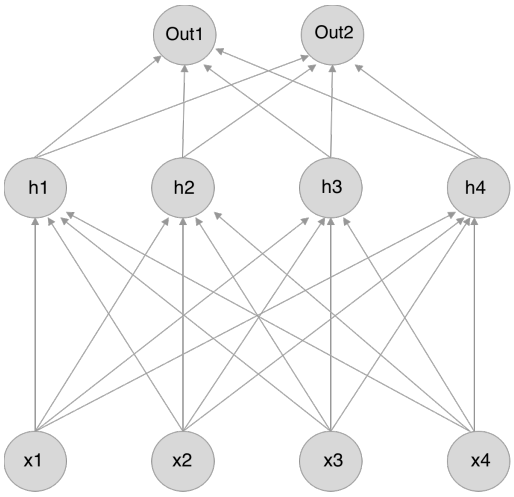

Fig. 2: This diagram illustrates the basic concept behind the fully connected layer by showing the last step of the architecture. In the diagram, x 1 through 4 represent the neurons of the pooling layer that preceded it, h1 through 4 represent the neurons of the fully connected layer, and Out 1 and 2 represent the final confidence values that will be outputed to the network.

are more complex features of the image such as shapes and objects. For example, for an image of a handwritten eight, 0.9025 could refer to the top and/or bottom circles of that eight.

These higher level features are the main advantage of the fully connected layer. Without the fully connected layer, the convolutional neural network would not be able to make a prediction as it would not be able to fully identify what objects and shapes are in an image. Without these shapes or objects, there is nothing to help identify what the image is. This is why every single architecture of a convolutional neural network contains a fully connected layer. However, the fully connected layer is not relegated to finding higher level features. Convolutional neural networks always end with the fully connected layer as it is able to take in all of the features that were found in the previous layer and output a final prediction in the form of a confidence value.

B. How Does It See An Image?

Although each layer had their own example, it is important to understand how the network works as a whole. Even though the previous section worked through many examples of each individual layer, it is easier to understand how the network works by walking through a full example. By walking through an example, it is easier to see the interactions between the layers and what the layers contribute to the network as a whole. In this example, a more defined architecture will be used rather than that of the previous section. For this example, let's say there is a convolutional neural network with the architecture: $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 2 \rightarrow FC \rightarrow RELU \rightarrow FC$, which is more concrete than the one used in the previous section as there are defined numbers of repetitions for certain layers.

Fig. 3: An advertisement for a Tesla Model X for reference.

For this example, let's say there is a collection of images that can be clasified as a car, truck, airplane, train, or bicycle. Out of this data set, the first image that will be fed in to the model is an advertisement for Tesla's Model X, which is seen in Figure 3. The model will first take in the image as input and break down the image into a photoreceptor array. In this array the values will range from 0 - 255 and

represent how light or dark the pixel is, 0 being white and 255 being black. Once the image has been fully pre-processed into this array, the image is ready to be analyzed by the network [7].

The array starts by entering the first combination of layers, $[CONV \rightarrow RELU \rightarrow POOL] * 2$. The convolutional layer will make the first pass and find all the low-level features of the image that it can by passing various filters onto the image. These low-level features range from edges to blobs and represent the image in its most basic form [9]. Once these low-level features are found, they are

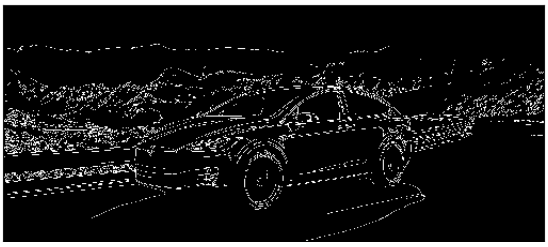

Fig. 4: This is how the convolutional layer would deconstruct the image in Figure 3. In this pass, the convolutional layer decided to detect all of the edges of Figure 3

then passed through a ReLu layer. The ReLu layer goes through each of these features and decides whether or not a feature should be used. Once the important features are decided, the information must pass through a pooling layer. This pooling layer distills each of the features further by deciding which parts of a feature are more important. For example, in Figure 3, the edges that make up the hood of the Model X are more important than the edges that make up the mountains behind it. After these important features are found, the output of the pooling layer is then fed into another convolutional layer and the whole process is repeated as the *2 in the architecture refers to the combination repeating two times.

After going through the first combination, the network is left with a lot of low-level features. However, in order to make the bigger classification, the network needs to find higher level features. Without higher level features, the network would not be able to classify what the image is. This problem is where the last few steps of the architecture come in, $FC \rightarrow RELU \rightarrow FC$. In this section of the architecture, all of the low-level features that were found by the previous layers are now being analyzed to become higher level features. Continuing with the Tesla Model X example, all of the important low-level features of the car that were returned by the last pooling layer are now being fed into a fully connected layer. The fully connected layer computes the higher level features of the image by piecing together each individual feature and analyzing them as a whole. Once the fully connected layer is finished computing the higher level features, the output is then fed into a ReLu layer which decides whether or not to use certain higher-level features. After sorting through the higher level features that are needed and those that are not, the output is then fed into the last step of the neural network, another fully connected layer. This layer performs the same function as it had before, but each of the values do not correspond to higher level features. Since this is the last step for the network, the fully connected layer, summarizes what the network has found into a set of confidence values. In this example, there would be five confidence values as there are five possible classifications for the image. If the network is correct and all sets of weights and biases for each neuron is optimal, then the highest confidence value in this example should be a car.

III. CNN-RNN

The CNN-RNN is one of the more recent algorithms for computer vision. This network was first proposed on June 17, 2014 by Darrel et al, and was called the Long-term Recurrent Convolutional Network [4]. In the paper, Darrel et al proposed

model where a convolutional neural network and a recurrent neural network were combined in order to caption images. Within the model, the convolutional neural network part would analyze the image and make a summary of the analysis. The recurrent network part would then take this summary and put it into words [4]. However, this network went unnoticed for a few years, until 2016, when Wang et al proposed a similar model named the CNN-RNN. There are many different types of CNN-RNNs architectures as the both parts of the network have their own different architectures. Ever since it was proposed, the CNN-RNN has started to gain traction as it has constantly proved to be an effective model for many image classification tasks.

A. Components

The network starts off by taking an image as input. A convolutional neural network then analyzes the image. After this analysis is complete, the convolutional neural network summarizes the image into sequential data. Sequential data is just data that is arranged in a way where bits of data that are relevant to each other are put next to each other. This data then acts as input for the recurrent neural network which generates a label for the image. This final label is what determines how the network has chosen to classify the image.

1) Convoltuional Neural Network: The job of the convolutional neural network is to break down the image into a format that the recurrent neural network can understand. In order to do this, the convolutional neural network must return a vector where different values correspond to different features of an image. In order to do this, many types of convolutional neural network architectures can be used, but it must end with a fully connected layer. However, this fully connected layer should not feed in the final prediction of the convolutional neural network as it would mess up the recurrent neural network. For example, if we trying to integrate the architecture $INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 2 \rightarrow FC \rightarrow RELU \rightarrow FC$ into a CNN-RNN. The first fully connected layer would be used rather than the second one as the second one would contain the confidence values for a given image, which have a very big range in values.

2) Recurrent Neural Network: If there was one word that could accurately describe a recurrent neural network, it would be a loop. A recurrent network, at its heart, is a network that continually performs the same task for each part of a given input [9]. For each pass, the weights and biases used are the ones that were calculated from the previous pass. For example, in Figure 5, the weights, and biases that were used in the first A pass would be used in the second A pass and so on. This continuously updates the weights and biases allowing the network to use what it had previously learned on to the new input.

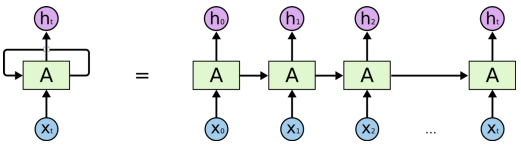

Fig. 5: This image illustrates the unraveling of a recurrent neural network and was made by Jianqiang Ma [20]

The type of recurrent network that the CNN-RNN uses is the long short-term memory network. Long short-term memory, LSTM for short, is a modified version of the recurrent neural network that was made for long-term dependencies [9]. In traditional recurrent neural networks, only the values from the previous pass are used to determine what the output of the current cell will be [11]. This limit presents a big problem as for longer inputs, more of the previous outputs are needed to make a good assessment. LSTMs fix this problem by being able to use information from earlier passes and choosing

what information to keep [11]. This ability provides an additional advantage as it fixes the **vanishing gradient** and **exploding gradient** problem that are apparent in recurrent neural networks.

In order to keep track of this historical data, LSTMs break down the information that is passed down from each state into two main variables: the *hidden state* and the *cell state* [9]. The hidden state acts like long term memory keeps track of the historical data through out the process, while the cell state acts like short term memory and keeps track of both recent data and parts of the historical data [14]. As Figure 6 shows, the hidden state is one long continuous chain that goes from the input to the output, while the cell state is constantly being changed.

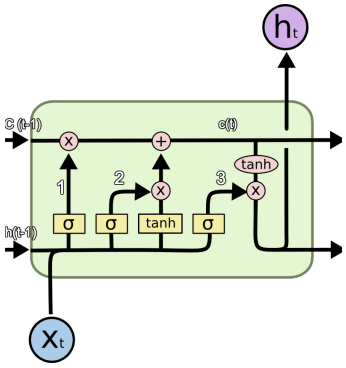

Fig. 6: This is a closer look as to what an LSTM cell would look like. h refers to the hidden state and c refers to the cell state [22]

For the LSTMs to calculate these hidden states and cell states, the cell has to perform a variety of complex functions. The LSTM first takes in the historical data with a START key at the beginning and an END key at the end. These keys tell the network when to start processing the input and when to stop. This data is then pushed into the LSTM cells. In order to understand LSTMs it is important to understand the cells themselves and how they work. LSTM cells consist of three main parts: *forget gate*, *input gate* and *output gate*.

a) Forget Gate: The forget gate is in charge of removing information from the cell state[11]. This gate looks at both the previous hidden state, and the new input and decides if any new information affects previous information. Figure 7 displays the basic structure of the forget gate.

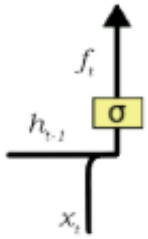

Fig. 7: A breakdown of the forget gate from Figure 6 [22]. In this figure, x_t refers to the input, h_{t-1} is the hidden state, σ is the sigmoid function and f_t is the output of the gate.

In order to determine whether or not a certain piece of information should be used, the forget gate takes in the newly found information, x_t, and the historical data,h_{t-1} and passes it through the sigmoid function. The sigmoid function then computes f_t, a **vector**, whose values range between 0 and 1 [4]. These numbers judge what information should be kept and what information should be deleted in the current state. These numbers are then multiplied to the cell state through **cross multiplication**.

b) Input Gate: The input gate adds the new information to the cell state. This gate takes in both the hidden state and the new input and performs two functions on it: the sigmoid function and the tanh function.

As Figure 8 shows, the first step is to perform the sigmoid function. In this case, the sigmoid function takes in the historical data, h_{t-1}, and the latest input, x_t as input and then returns a vector, i_t whose values

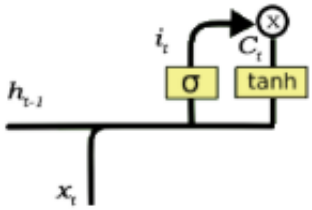

Fig. 8: A better look at the Input gate of the LSTM. The x represents multiplication that is done between C_t and i_t

range from 0 to 1, and dictates whether or not a piece of information needs to be added [9]. The second part of this gate is to compute the **tanh function**. This function takes in the same input as the sigmoid function and determines whether or not a piece of information can be added to the cell state. The tanh function computes another vector, C_t whose values range from -1 to 1. After C_t is computed, it is then cross-multiplied with i_t[14]. These vectors are then added to the cell state-after the cell state is modified by the forget gate.

c) Output Gate: The output gate is in charge of what determining what the cell should output [4]. In order to do this, the gate must look at both the current state and the hidden state. This output is then given to the next pass as well as outputted by the cell. In other words, the output gate does not affect the cell state at all.

As Figure 9 shows, the output gate has three main steps. The first step is to take the current cell state and apply the tanh function to it. This step is very important as it simplifies the values of the cell state vector to only range from -1 to 1. The second step is to then take the hidden state and the input and feed these two into the sigmoid function to produce o_t. o_t determines the final values that should be represented into the output. The last step is to cross multiply o_t to the output of the tanh function. this last vector is what is outputted out of the network

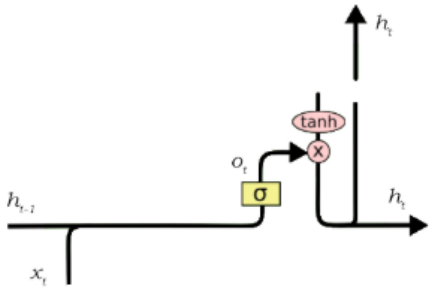

Fig. 9: A closeup look at the output gate of the LSTM shown in Figure 6. In this figure, x_t is the new input, h_{t-1} is the previous hidden state, σ is the sigmoid function, tanh is the tanh activation function, the $\times$ is the cross multiplication, and h_t is the final output of the gate and the LSTM cell.

as well as fed into the next iteration.

B. How Does It See An Image?

The CNN-RNN is a very complex network, so in order to understand how the network works, it helps to have a step by step walkthrough of the network as it generates a label for an image. Before getting into the example, it must be said there are a multitude of different architectures that exist for the CNN-RNN.

For this example, the CNN-RNN model will have this structure $INPUT \rightarrow CONV \rightarrow RELU \rightarrow FC \rightarrow LSTM$. In this representation, everything from the INPUT to the FC is the convolutional neural network part and the LSTM is the recurrent neural network part. INPUT refers to the image that is given to the network in the form of a photoreceptor array, CONV refers to the convolutional layer, RELU refers to the rectifier layer, FC refers to the fully connected layer, and LSTM refers to the LSTM layer. Let's say that we have trained this CNN-RNN model and now want to test the network on a collection of images. Out of this collection the first image that will be fed into the network is a

picture of a woman playing frisbee, shown in Figure 10a.

(a) Initial Image

(b) Focusing on the woman's body

(c) Focusing on the frisbee

Fig. 10: Differnet versions of the image that will be fed into the example model. The image was taken from Xu et al [28]. **a)** the initial image for refrence. The final label is a woman playing with a frisbee. **b)** This shows what the LSTM is focusing on to make the woman part of the final label **c)** A refrence image to show how the LSTM focuses on the frisbee part of the image.

First, the image has to pass through the convolutional neural network part of the model. This part will first break down the image into a photoreceptor array, which will make the image readable for this section of the network. This array is then fed into a convolutional layer, which will break down the image into low level features, such as the edges of the arm and the frisbee). After all of the low level features are found, the ReLu layer goes through each of them and decides whether or not a certain feature is important or not. The convolutional neural network section of the CNN-RNN, then ends with a fully connected layer, which deciphers the higher level features of the image (i.e. shapes). However, unlike the last fully connected layer of a convolutional neural network, this layer does not output a prediction [4]. Instead, features that are grabbed by the fully connected layer act as a summary to the entire image. Before these features can be put into the recurrent network part of the CNN-RNN, they must first be **encoded** into sequential data. In order to do this, the fully connected layer puts features that correspond to each other next to each other. For example, in Figure 10a, the fully connected layer may have found the woman's arm and the woman's body to be higher level features. Because the two features are connected, they will appear next to each other in the sequential data [2]. While encoding this data, the network will put a START key at the beginning and an END key at the end to guide the recurrent network.

Once the convolutional neural network section encodes its finding into sequential data, the recurrent neural network part then decodes it to make the final prediction. The LSTM will first take the sequential data and go one by one to generate a word for the final caption until it hits the END key. During one pass the LSTM will first start off by taking in the hidden state and the cell state from the previous pass (if it is the START key, it starts off with just the initial input). This information corresponds to

certain parts of an image, as seen in Figure 10b and Figure 10c. The LSTM will first take these three inputs and put them through the forget gate in order to decide whether or not to use the cell state from the previous pass. After the forget gate, the LSTM then takes the previous hidden state and the input and puts it into the input gate. The input gate starts to decide what the current pass' cell state will be. Finally, the hidden state and the input are put through the output gate. The output gate decides what word that this pass of the LSTM will output as well as what the hidden and cell state for this pass is. Once the word is outputted, the cell state and the hidden state are fed into the next pass of the LSTM. This process is repeated until the END key is reached.

After finishing all of the passes, and assuming that the weights and biases of the convolutional neural network section is optimal, the model should output a caption that can range from vague, such as "a woman playing frisbee", to more specific,"a woman in a green suit playing with a red frisbee". However, as long as the network identified both the frisbee and the woman, the network has correctly classified this image.

IV. Analysis

After seeing how both networks work, it is time to put the networks to the test. However, before delving into the experiments that were conducted it is important to acknowledge the advantages and disadvantages of each network. These advantages and disadvantages are just as important as the results of the experiments when deciding which algorithm is the best one. Because of this fact, this section starts off with the advantages and disadvantages section first as it gives a better frame of reference as to where the strengths and weaknesses of each algorithm lie. This section then goes into the results of the various experiments that tested both networks.

A. Advantages & Disadvantages of Each Network

1) Convolutional Neural Networks:

a) Advantages: One of the main advantages of ConvNets is that the input of the network is assumed to be an image, or in an image like format [9]. This assumption provides a huge advantage, as an image does not have to go through a pre-processing stage, where the image would have had to been augmented to fit into the network. With this assumption, ConvNets are able to decipher images faster as it can also perform many image specific tasks. Many of the layers of the ConvNet, such as the max pooling and convolution layer, are also made to work with this assumption. With these adaptations, many people have said that the ConvNet was made for image processing [3].

The other main advantage of using a ConvNet for computer vision is that it saves time and memory. Training time and memory allocation have become commodities when it comes to deep learning algorithms [3]. Training time is used to measure efficiency as faster algorithms are able to be more commercially used for their speed [12]. After all, for real-world applications, an algorithm that takes too much time to be trained would not be able to efficiently analyze large amounts of data. On the other hand, memory allocation has become a factor in measuring a models efficiency, as these algorithms require a lot of storage for all of the continually changing weights, biases and other outputs of functions.

b) Disadvantages: One of the main problems with the ConvNet is that the receptive field is fixed [29]. A fixed receptive field means that the network always has the same input size. While this fixed input size, may work with various images, the problem arises when the network is asked to find smaller details in a large image. When this task is given, the network shrinks the image through various convolutions, which can make the image

lose a lot of important details [29]. Without these detail, the convolutional neural network could fail at the task at hand. For example, let's say you feed an image of the famous painting *The Death of Socrates* by Jacques-Louis David, shown in Figure 11, and asked the network to identify all of the faces in the painting.

Fig. 11: A picture of *The Death of Socrates* by the late 18th century painter, Jacques-Louis David [5].

In an attempt to cut down the image to focus on what the network thinks are important details, the network may cut out Plato who sits at the foot of Socrates' bed as it may decide that Plato does not have the qualities of a face. By cutting out Plato, the network does not complete it's task adequately.

Another problem with ConvNets is that they do not accurately represent the human visual cortex. ConvNets are based on Fukushima's "Neocognitron" which was a deep learning algorithm that was based on Hubel and Wiesel's theory on how the human visual system worked [7]. However, modern research shows that a human's perception may not only be entirely based on the "feed forward" processing that Hubel and Wiesel theorized. Hubel and Wiesel's theory stated that the human visual system is made up of a topological map that feeds forward to various columns of neurons in the brain until an object is deciphered [6]. This misrepresentation is a problem as it means that there is an algorithm that could more accurately mimic the human visual system. An algorithm that could accurately mimic this system would be a better computer vision algorithm as it could perform various computer vision problems more efficiently.

2) CNN-RNN:

a) Advantages: There are many advantages to using the CNN-RNN. For example, one of the biggest advantages for the CNN-RNN is that it is transferable [10]. Transferable refers to the fact that the CNN-RNN works with any different ConvNet architecture or recurrent neural network architecture, as long as the ConvNet can feed into the latter. This flexibility is a huge advantage as it opens a whole new door for mixing and matching different architectures in order to build the best comptuer vision algorithm [10].

Another advantage is that the labels that are outputted do not have to have a fixed length. This is a huge advantage as it allows the network to make more specific classifications and more accurate labels. These variable label sizes also allow the network to learn different distinguishing features from other images and make connections with other labels. The more accurate the label that the network outputs is, the better the network becomes at classifying images.

b) Disadvantages: There are two glaring disadvantages that effect the CNN-RNN. One of the problems of the CNN-RNN is that it has trouble detecting smaller images [13]. Since many small objects don't have many defining features or these features are not very present in the image, they often go unnoticed by either the convolutional neural network part or the recurrent neural network part. For some images, overlooking these small segments of the image can prove to be fatal as it could lead to more important features of the image being completely ignored.

The other disadvantage of the CNN-RNN is that the network is only able to produce labels [10]. In

certain cases, the algorithm may try to label everything that it could possibly can. This is troublesome as many noisy images may have paragraphs for labels which do not give an accurate representation of the image. In the computer vision community, the preferred method for image classification is to draw a box around what the network is trying to identify and labeling the box rather than printing out a label for the image. For example the YOLO (You Only Look Once) algorithm for object detection, a modified convolutional neural network, has become very popular in the computer vision community as it emphasizes the object that it detects by drawing said boxes [16]. This preference negatively effects the CNN-RNN because the CNN-RNN can only output captions, limiting what the network can and can't output.

B. Results

1) Procedure/Methodology: All computer vision experiments have to follow the same procedure and methodology [8]. The only things that can differ from each study are the networks being tested and the dataset being used. The first step is to lock down a dataset or multiple datasets to use for the experiment. Once a dataset is found, the dataset is then split into training and testing sets. This split allows researchers to keep parts of the dataset to help make the model and parts of the dataset that help to assess the model. The split does not have to correspond to anything and can be completely random. After the dataset is split, the models are then trained on the training set. Once the models are trained they are ready to be tested. During testing, a researcher keeps track of how many images were correctly classified and how many images were incorrectly classified. These results are then put into a confusion matrix which keeps track of what the model labeled each image as and whether or not that label is correct. The results of the confusion matrix can be analyzed with two different methods: Top-N error rates and statistical analyses.

Top-N error rates are calculated by finding the amount of images that were were not classified correctly for N-amount of classes [2]. For example, let's say that a convolutional neural network was being assessed for whether or not it could classify people. When a picture of a person is fed into the model, the confidence values for the network were: dog: 0.74, cat: 0.52, human:0.68, fish: 0.16, elephant: 0.43 and horse: 0.34. In this example, while calculating the Top-N error rates, top-5 would say that the image was correctly classified as human appears in the top-5 values, while top-1 would say that the image was missclassified as the highest value is not for human. Although there are many values that can be used for N, the most popular values are 1 and 5 as they narrow down a wide array of classes into a small amount. The lower the Top-N error rate is, the better the model is as a lower error rate means that more images were correctly identified and vice versa.

Statistical analyses of the models is based upon four key values: true positives,true negatives, false positives, and false negative [8]. True positives are times where the model correctly identified what the image was, true negatives are times where the model correctly identified what the image was not, false positives are times where the model incorrectly identified predictions for certain classes and false negatives are where the model incorrectly rejected a label for a class. Statistical analyses uses these values in order to calculate a model's accuracy in terms of score. The two most popular scores are the accuracy score and the F1 score.

The accuracy score measures accuracy by comparing the total correct answers to all of the guesses made. In terms of the key values, accuracy is

calculated using the following equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

In the equation above, TP refers to true positive, TN refers to true negative, FP refers to false positive and FN refers to false negative. As the equation shows, accuracy takes all of the correct guesses and misses made by the algorithm and divides it by the total guesses. As the equation shows, a higher accuracy score means that a model is doing well as more correct guesses are being made in relation to the total amount of guesses.

The F1 score is another way to measure accuracy and is based around two other scores, precision and recall [9]. Precision refers to the percentage of relevant results [8]. Precision is calculated using the following equation:

$$Precision = \frac{TP}{TP + FP}$$

As the equation shows, precision finds relevancy by looking at the amount of correctly labeled results, true positive, and comparing it to the total correct predictions, TP+FP. Recall refers to the amount of actual positives the model found [8]. In order to find recall, the following equation is used:

$$Recall = \frac{TP}{TP + FN}$$

In the equation, the sum of the true positives and the false negatives refer to total amount of actual positives found by the network. Precision and recall are given more weight when measuring different things. However, since the two can be used interchangeably, the F1 score is a better metric [8].

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

As the equation above shows, the F1 score is calculated by taking the weighted average of both precision and recall. The F1 score is a better representation of how the model performs than precision and recall are as the F1 score takes both false positives and false negatives into account [8]. A model should strive to achieve a higher F1 score as a higher F1 score means that the model is more accurate when it comes to labeling images. F1 scores are used when there is an uneven amount of labels in a given data set.

2) Data: In order to see which computer vision algorithm is the best for image classification, three research groups, Guo et al, Chen et al and Wang et al, decided to test the two networks against each other.

TABLE I: Accuracy score of each network found by Guo et al and Chen et al [2] [10].

Network	**Accuracy Score**
CNN [2]	63.21%
CNN [10]	76.01%
CNN-RNN [2]	65.07%
CNN-RNN [10]	82%

Table I displays the results of Guo et al's experiment. Guo et al took a convolutional neural network and a CNN-RNN and used the ImageNet 2010 dataset to train and test each network. After testing, Guo et al calculated the accuracy score for each network. As Table I shows, the CNN-RNN was more accurate than the convolutional neural network, referred to as a CNN in the table, by 4.99%.

TABLE II: Top-1 and top-5 error rates of each function found by Chen et al [2].

Network	**Top-1**	**Top-5**
CNN	42.71%	20.01%
CNN-RNN	41.85%	19.15%

Another experiment to test the two different networks was conducted by Chen et al. In this experiment, Chen et al took the CNN-RNN and compared it to the AlexNet, a special architecture of the convolutoinal neural network, by using a

variety of datasets to train and test the two models [2] [15]. Table II shows the results of the two networks using the ImageNet Large-Scale Visual Recognition Challenge 2012 dataset and Table I shows the results using the MIT Indoor dataset. As both tables show, the CNN-RNN was the better network than the convolutional neural network as it was 1.86 % more accurate and had approximately a 1% less top-1 and top-5 error rate. This means that the CNN-RNN was better at classifiying the images as more images where correctly labeled than the convolutional neural network. Although these error rates have a very small difference, the accuracy scores show that the CNN-RNN was better overall.

TABLE III: Overall recall, precision and F1 scores found by Wang et al [13].

Network	**F1**	**Recall**	**Precision**
CNN	65.7%	61.3%	68.5%
CNN-RNN	67.8%	66.4%	69.2%

The last experiment was conducted by Wang et al. Wang et al took the CNN-RNN and the convolutional part of the CNN-RNN and tested it on the NUS-WIDE dataset [13]. After testing, Wang et al calculated the F1 score, the recall score and the precision score for each network. As Table III shows, the CNN-RNN was the better algorithm. Even though both networks had similar precision scores, the CNN-RNN was greater by 0.7 %, the CNN-RNN had a a 2.1% better F1 score and a 5.1% better recall score.

V. Conclusion

Although there is much debate in the computer vision community as to what the optimal algorithm is for image classification, the data above shows that the CNN-RNN is more accurate than the convolutional neural network. This increase in accuracy may be caused by the recurrent neural network of the CNN-RNN.

When the CNN-RNN was proposed, the idea behind it was that by linking a convolutional neural network to a recurrent neural network, the convolutional neural network would do the bulk of the analysis, while the recurrent neural network would put this analysis into words [4]. If this were the case, then both networks would have had scores with a difference of at most 1 %. Although this is the case for the top-1 error, top-5 error and the precision score for each network, the recall score, f1 score and accuracy score of each network prove otherwise. For all three of the scores (recall, F1, and accuracy), the CNN-RNN scored higher than the convolutional network. This increase would have to be caused by the recurrent neural network part of the CNN-RNN as in the Wang et al and Chen et al's experiments, convolutional neural networks were just the convolutional neural network part of the CNN-RNN[2] [13]. The increase in accuracy could be caused by the recurrent network fine tuning the summaries given by the convolutional neural network part. As Figures 10b and 10c show, the recurrent network is capable of focusing on different aspects of an image. This ability allows the recurrent network to further analyze parts of the image before labeling the image.

Whether or not the recurrent neural network part of the algorithm is the reason why the CNN-RNN outperformed the convolutional neural network is uncertain. Further research is needed to pinpoint the exact reason. It is important to pinpoint this reason before the CNN-RNN is ready to be implemented for real world applications.

VI. Glossary

Backpropagation- how the network analyzes its weights and biases. Back propagation goes backwards and computes the gradient for every activation function that is called. This gradient determines how effective the current weights and biases.

Bias- A value that corresponds to how far off a certain prediction is from the real value. It is used when calculating activation functions.

Cross Multiplication-The process in which two matrices are multiplied together to produce another matrix.

Exploding Gradient-A problem where the gradient computed shows that there is too large of an increase in the function.

Gradient- A vector whose components are the partial derivatives of a function. These partial derivatives tell the network how much a change in the weights or bias affects the network as a whole

GPU processing- The process where all of the computations are run through the GPU (Graphics Processing Unit) instead of the CPU. Most processes run through the CPU (Central Processing Unit), however, since deep learning algorithms are massive, a CPU takes a long time to do all of the processes needed. To fix this, researchers realized that a GPU could be manipulated to act like a better/faster CPU.

Sigmoid Function- A function that only ranges from 0 to 1, but has a higher rate of change than the ReLu function.

Tanh Function- The tanh function is a special type of activation function that uses the hyperbolic tangent function to calculate a neuron's activation. Because it uses this function, the range of the function can only be from -1 to 1

Vanishing Gradient-A problem where the gradient computed is too small, causing nothing to be updated.

Vector- An array with only one dimension.

Weights- A value or values that decide the strength of the connection. It is used when calculating activation functions.

References

[1] Abien Fred M. Agarap. Deep learning using rectified linear units (relu).

This paper presented a new way in which the ReLu function could be used. Traditionally, ReLu functions where used as activation functions, this paper sees if a ReLu function could be used for classification instead of an activation function. In order to get the results, Aparap compared the modified CNN to a traditional CNN as it classified the various digits in the MNIST dataset. Although the paper does provide interesting insight into how the ReLu function could be used, the only useful information for me was the indepth explanation of the ReLu function. Abien Fred M. Agarap is a lead research scientist at Senti Techlabs.

[2] Yushi Chen, Xiao Liu, Bing Shuai, Bing Wang, Gang Wang, Xingxing Wang, and Zhen Zuo. Convolutional recurrent neural networks: Learning spatial dependencies for image representation. *Computer Vision Foundation*, 2015.

This paper proposed a model where a Convoluitional Neural network fed into a recurrent neural network. Even though this paper did give me a great insight into how the network was designed, its explanation on LSTMs were extremely useful as it made it very clear. The paper detailed an experiment that tested the proposed model to the Alex-Net, a very popular pretrained ConvNet. In order to test the two networks, both models were trained and tested on three different data sets, ILSVRC 2012, SUN397 and MIT indoor. They found that the proposed model had a greater accuracy rate than the Alex-Net.

[3] Dan Ciresan, Ueli Meier, and Jürgen Schiduber. Multi-column deep neural networks for image classification. Technical report, Dalle Molle Institute for Artificial Intelligence, 2012.

This article details the use of Convolutional Neural Networks as it out performs usual neural networks on various visual datasets. The authors assert this statement by testing the neural networks on a variety of popular datasets, such as the MNIST and NORB datasets, and then calculating the classification score. The scores were then compared to each other. What the researchers found was that the multi-colimn ConvNet used was almost on par with a human. These results not only show me how powerful ConvNets are, but also how these networks are trained and measured, as well as an explanation behind the datsets. These datasets are almost always brought up when talking about computer vision and the explanation of these datasets allow me to see what a proposed

model is being trained for. Jurgen Schmidhuber is the co-director of the Dalle Molle Institute of AI Research, Dan Ciresean is a senior reseracher at the Dalle Molle Institute for Artificial Intelligence and Ueli Meier is a reseracher at IDSIA.

[4] Trevor Darrell, Jeff Donahue, Sergio Guadarrama, Lisa Anne Hendricks, Marcus Rohrbach, Kate Saenko, and Subhashini Venugopalan. Long-term recurrent convolutional networks for visual recognition and description. *Computer Vision Foundation*, 2014.

This paper was one of the first papers to ever introduce a model that conjoined a ConvNet to a recurrent neural network.The paper then focused on how effective such a model sould be for image labeling, and video classification. In order to test this new model, the team of researchers trained and tested the model on the UCF-101 and an unnamed dataset of videos. The paper found that the proposed model had a higher recall score than any other model that was tested on the datasets. The results of this paper helped give data to back up the assertion that the CNN-RNN is better than the ConvNet. Trevor Darrell is a professor at the University of California, Berkeley and is the co-Director of the Berkeley Artificial Intelligence Research Lab, Jeff Donahue is a research scientist at DeepMind, Sergio Guadarrama is a software engineer at Google AI, Lisa Anne Hendricks is a graduate student researcher at the University of California, Berkeley who is studying with Prof. Trevor Darrell, Marcus Rohrbach is a research scientist at Facebook AI, Kate Saenko is an associate professor at Boston University, and Subhashini Venugopalan is pursuing her Ph.D in computer vision at the University of Texas at Austin.

[5] Jacques-Louis David. The death of socrates, 1787.

[6] Kate Fehlhaber. Hubel and wiesel and the neural basis of visual perception. *Knowing Neurons*, 2014.

[7] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980.

This ground breaking article was one of the first instances of the idea of a convolutional neural network. Modeling his previous cognitron model after the biological model for human vision, Fukushima creates, what we now know as, a convolutional neural network. This article was helpful to me as it illustrated the foundations of how the first convolutional neural networks were built. Many of the same concepts in the Neocognitron are still used in modern ConvNets, such as the S-cell and C-cell type nodes. The article uses mathematical proofs as well as data gathered from computer simulations to show the effectiveness of the model. Kunihiko Fukushima is a senior research scientist at Fuzzy Logic Systems.

[8] Aurelien Geron. *Hands-On Machine LEarning wiht Scikit-Learn & TensorFlow*. O'Reilly, 2017.

This book was a great introduction for my research. It layed out various classification models before delving into deep learning algorithms and how to implement them using TensorFlow, a machine learning library. By walking through various examples and excercises, the book teaches you each of the concepts that it presents. Although it did explain convolutional neural networks and recurrent neural networks, the main asset of this book was the way it outlined how experiments for deep leanring should run and how to assess each model. This helped me to write the results section of this paper. Aurelion Geron led Youtube's image classification team and is the founder of Kiwisoft, a machine learing consulting firm.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Chapter 9 and Part I of this book provides with specific information for Convolutional Neural Networks. While Part I deals with the mathematical aspects of deep learning, it is essential knowledge in order to read Chapter 9 which deals specifically in Convolutional Neural Networks. Chapter 9 is useful as it shows the mathematical concept of a convolution while also describing the advantages of a convolutional neural network. This book displays its arguments by using textbook language as well as various statistics. This book was written by: Ian Goodfellow, a research scientist at Google Brain, Yohua Bengio, head of Montreal Institue of Learning Algorithms and Aaron Courville, an assistant professor at the University of Montreal.

[10] Yanming Guo, Yu Liu, Erwin M. Bakker, Yuanhao Guo, and Michael S. Lew. Cnn-rnn: a large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, 2018.

This paper dealt with training an algorithm that combined the convolutional neural network and the recurrent neural network(CNN-RNN). In this paper, the reseracher ran various versions of the ConvNet and a CNN-RNN to see which network was better at detecting coarse and fine labels. Coarse labels were defined as the superclass of the object and the fine labels were defined as the exact class of the object. For example, dog would be considered as a coarse label and golden retriever would be classified as a fine label. This data was gathered by training these networks on a subset of the 2010 ImageNet dataset, the CIFAR 100 dataset and the 2012 ImageNet dataset. The authors than calculated the accuracy scores of each network, only to find that the CNN-RNN was more accurate than the ConvNet for the ImageNet 2010 dataset, the ImageNet 2012 dataset and the CIFAR 100 dataset. The results of this paper was really helpful

for my understanding of the CNN-RNN and provided the data for the analysis section/Yanming Guo is a researcher at Leiden Institute of Advanced Computer Science(LIACS) in the Netherlands and specializes in computer vision applications. Erwin Bakker is the codirector of the LIACS, Michael Lew is a researcher at the LIACS. and is a professor at Leiden University. Yu Liu is researcher at LIACS, recieved a Ph.D from Leiden University and reseraches computer vision and image recognition. Yuanhao Guo is a teaching assistant at Leiden University and has conducted reasearch in the field of computer vision under Profesor Michael Lew

[11] Sepp Hochreiter and Jurgen Schmidhuber. Long short term memory. *Neural Computation*, 1997.

Long Short Term Memory by Hochreiter and Schmidhuber is a ground breaking paper for the deep learning community. In this paper, Hochreiter and Schmidhuber propose a new type of recurrent neural network that quickly changes it's sets of weights. In order to see if the proposed model would work, the two researchers ran seven experiments. The first experiment saw tested if the model could predict the next words in a sentence by using the embedded Rebar Grammar. The other six of these were modifying the input was given. For example experiments 2-3 dealt with adding noise to the input. The paper found that the LSTM was able to handle noise and was able to learn without parameter fine tuning. This landmark paper has become the basis for many other algorithms and papers that are written today. However, for me, the paper's main utility was it's explanation of the vanishing/exploding gradient problem and of the LSTM itself. Sepp Hochreiter is currently the head of the Institute for machine learning at the Johannes Kepler University at Linz and Jurgen Schmidhuber is the co-director of the Dalle Molle Institute for Artificial Intelligence.

[12] Jeremy Howard et al. Deep learning for coders, 2018. https://github.com/fastai/fastai.

This series of lectures and lecture notes walked through many types of deep learning techniques, as well as applications of these techniques. For this paper, the most important lessons are Lesson 2, which focused on convolutional neural networks, Lesson 6 which focused on Recurrent Neural Networks, Lesson 7, which focuesd on the different convolutional Neural network architecutres and lesson 8, which focused on reading deep learning papers and how to navigate through the different parts of code that may come up when reading these papers. Lesson 2 showed me how each of the layers of a ConvNet interact by walking through both theorems and implementations. Jeremey Howard is the founder and head of fast.ai, a company that helps teach deep learning.

[13] Chang Huang, Zhiheng Huang, Junhua Mao, Jiang Wang, and Yi Yang. Cnn-rnn: A unified framework for multi-label image classification. *Computer Vision Foundation*, 2016.

This paper made the CNN-RNN famous. In this paper, Wang et al proposed a model where a ConvNet was combined with a recurrent neural network. To test the effectiveness of this network, Wang et al trained and tested six control models and a CNN-RNN on the NUS-WIDE dataset, the Microsoft COCO dataset and the PASCAL VOC 2007 dataset, and computed their precision and recall scores. The experiment showed that the CNN-RNN had the highest precision and recall scores and was therefore the most effective network out of the seven for image classification. Jiang Wang is a researcher at Google Research, Junhua Mao is a researcher and software engineer at Waymo Inc, a startup specializing in self driving cars, Zhiheng Huang is an AI researcher at Tencent, a chinese investment company, and Chang Huang is the cofounder of Horizon Robotics.

[14] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy Blog*.

Even though this is a blog post, it is still made by one of the lead researchers in computer vision, Andrej Karpathy. Andrej Karpath is a director of AutoPIlot Vision at Tesla and has written multiple papers on neural networks. Because it is written by one of the biggest researchers in computer vision, this article still gives some useful information about Recurrent Neural Networks. The article provides an in-depth look into the various uses of recurrent neural networks as well as how they work in each case. This helped me a lot as it gave me a basic understanding of LSTMs and recurrent neural networks, before I started to take a deep dive into the research papers and articles on the subject. However, it is still a blog post, so the credibility of the information still has to be questioned.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks.

In this paper, Alex and his team of researchers trained a special convoltuional neural network, later known as the AlexNet, on the datset from the ImageNet Large-Scale Visual Recogntion Challenge(ILSVRC). The AlexNet scored a Top-1 error of 37.5 percent and Top-5 error of 17.0 percent. This paper really helped me see how convolutoinal neural networks work. It also allowed me to analyze the work from Chen et al, as the convolutional neura network that Chen et al used was the AlexNet. Alex Krizhevsky is one of the most accredited researchers when it comes to the field of computer vision who last worked at the University of Toronto, Ilya Suskever is a research director at Open AI, and Geoffery E Hinton is a reseracher of AI at both Google and the University of

Toronto.

[16] Fred Lambert. Watch a tesla drive in paris through the eyes of autopilot. *Electrek*, 2018.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropogation applied to handwritten zip code recogntion. *MIT*, 1989.

This was another paper written by Yann LeCun. This paper looked at the effects of backpropagation on Convolutional Neural Networks. Backpropagation had strenghtened the CNN's effectiveness in identifying handwritten digits. In order measure the effectiveness of this method, the model was run 23 times on the training test and then tested. After testing, the researchers measured the models means squared error. Because the means squared error and the amount of misclassified images were low, many modern CNNs employ backpropogation in their models in order to perform their intended task better. This paper served as an explantion of how backpropagation works, and why it's useful. Yann LeCun is the director of AI Researcher at Facebook and has been instrumental in the development of CNNs.

[18] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. 1998.

This is one of the earliest uses of Convolutional Neural Networks, behind Neognitron, the only difference being that they utlized gradient descent. In this paper, Yann LeCun, a key researcher in the field of computer vision, and his team use a convolutional neural network to identify handwritten digits. They found that Convolutional Neural Networks outperformed previous methods for computer vision. This paper showed how gradient descent and backpropagation really strengthen a convolutional neural network by updating the weights. With this paper, I was able to learn about the gradient descent algorithm, how it is applied, and how effective it is when paired with the back propagation.

[19] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Cs231n: Convolutional neural networks for visual recognition 2016.

These series of lecture notes were written by the professors and displayed different types of convolutional Neural net architectures as well as the different layers of each architecture. Module 2 was more important to me than Module 1. The only part of Module 1 that I used was the backpropogation derivation. Module 2 dealt with Convolutional Neural network archetectures and layers. By breaking down the different architectures and layers that exist for a convolutional neural network, I was able to understand everything that there was to know. This coures showed how modern neural networks are constructed. Fei-Fei Li is the director of the Stanford Vision Lab, Andrej Karpathy is a director of AutoPilot Vision at Tesla and Justin Johnson is a research scientist at Facebook AI Research

[20] Jianqiang Ma. All of recurrent neural networks. *Medium*, 2016.

[21] Michael Nielson. *Neural Networks and Deep Learning*. Determination Press, 2015. https://neuralnetworksanddeeplearning.com/.

This book was a boiled down version of how neural networks. The most important chapters of this book were chapters 1, 2, 5 and 6. Chapter 1 walked through an application of a convolutional neural network, chapter 2 helped me understand the backpropagation algorithm, chapter 5 showed me the weaknesses of neural networks, and chapter 6 explained the concepts that were used for Chapter 1. All in all, the book helped me solidify my understanding of basic neural networks and allowed me to see how convolutional neural networks work. Michael Nielson is a research fellow at Y Combinator Research and has written many books and articles on machine learning and quantum computing.

[22] Christopher Olah. Understanding lstm networks. *colah's Blog*, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[23] Barnabas Poczos. Convolutional neural networks.

This article is part of a series of lecture notes from the Introduction to Machine Learning course at Carnegie Mellon University. The course is taught by Barnabas Poczos is an assosiate professor at Carnegie Mellon University. This specific lecture note focuses on convolutional neural networks and gave me a "bare bones" outline of the basic ideas of a ConvNet. This is helpful as it provides a quick refrence to certain concepts such as convolutions and the main ideas of each implementation.

[24] Vadim V. Romanuke. Appropriate number of standard 2×2 max pooling layers and their allocation in convolutional neural networks for diverse and heterogeneous datasets. *De Gruyter*.

This article went into the max pooling layer and when and how to apply it. The most helpful part of this article was the explanation of thelayer, the equations behind the layer and the layer's importance. In order to get this information, the author made varying ConvNets with different ratios of Max pooling layers to other layers and then tested it on the CIFAR-10, NORB, and EEACL26 datasets. The performance is then measured by the author's special metric as error rates would not provide a good representation of the effectiveness of the layers. Vadim V. Romanuke has written multiple papers on optimizing ConvNets.

[25] Jurgen Schmidhuber. Deep learning in neural networks: An overview. *The Swiss AI Lab IDSIA*, 2014.

This paper was made in order to track which researchers deserved credit for certain contributions. While also explaining why they deserve credit, the paper acts as an encyclopedia of many deep learning papers. This paper

allowed me to find various papers on convolutional neural networks and LSTMs and provided brief overviews on these papers as well. These brief overviews gave me a good understanding as to whether or not the article would have certain bits of information that I was looking for. However, this info Jurgen Schmidhuber is the co-director of the Dalle Molle Institute of AI Research.

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 2015.

This paper proposes a new type of Convolutional neural network based solely on convolutions. The paper found that taking out the max pooling layer of a convolutional neural network actually decreased the classification error by 2.6as it provided an argument against having a max pooling layer at all. Since the evidence is there, many are wondering whether this paper marks the end of the max pooling layer. Jost Tobias Springenberg works for the Machine Learning Lab in the University of Freiburg andhas published many papers analyzing different supervised and unsupervised machine learning models. Alexey Dosovitskiy is a researcher at Intel Labs in Munich and specializes in Computer Vision. Thomos Brox is a professor for pattern recognition and image processing and the head of the computer vision group at the University of Freiburg. Martin Riedmiller is a rearch scientist at Google's DeepMind.

[27] Rob Verger. Facebook used billions of hashtagged instagram photos to train its ai. *Popular Science*, 2018.

[28] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, 2015.

[29] Dong-Qing Zhang. Image recognition using scale recurrent neural networks. *ImaginationAI LLC*.

This paper was one of the only sources that attempted to use a recurrent network(rather than a CNN-RNN) for image classification. Although the recurrent network did have elements of a ConvNet, it is still considered to be a recurrent network. This paper offered an insight into the problems of ConvNets as well as the effectiveness of an RNN for image classification. In order to test the effectiveness of the network, Dong-Qing Zhang trained 4 different algorithms (two ConvNets, and two RNNs) on the ImageNet-1K dataset. He then calculated each of the networks Top-1 error rate. Although the results from this paper were interesting, the only part of this paper that helped was the analysis of the convolutional neural network at the beginning of the paper. This analysis really helped me gather data for the advantages and disadvantages section of my paper as Zhang's paper highlighted the ConvNet's strengths and weaknesses. Dong-Qing Zhang is currently a Ph. D. student at Columbia University, who has written extensively on Artifical Intelligence and Computer vision.

[30] Marrian Zhou. Amazon go expansion continues with store in chicago. *C Net*, 2018.

2. Fieldwork

2.1. Fieldwork Questions

For the second semester, I worked on a experimental project. To help me through the process, my guiding questions were:

(1) What architecture is best for each model for the given task?
(2) How do you implement these algorithms?
(3) What is the best way to implement these algorithms?
    (a) What is Keras?
        (i) How does it work?
        (ii) Pros & Cons
    (b) What is TensorFlow?
        (i) How does it work?
        (ii) Pros & Cons
(4) How does one assess these networks?
(5) What is the difference between a multi class and a multi labeled dataset?
(6) How does each model do on a multi class dataset?
(7) How does each model do on a multi labeled dataset?
(8) How can these models be used in the real world?

2.2. Mentors & Fieldwork Sites

2.2.1. Mentors

Klint Qinami
35 Olden Street
Princeton, NJ 08540
kqinami@princeton.edu

Background Information: Klint Qinami is currently pursuing his Ph.D. at Princeton under Professor Olga Russakovsky and is currently a part of the Princeton Visual AI Lab. Klint studied computer science at Columbia University and is well versed in TensorFlow With this experience, Klint would be able to help me with any programming problems that I may encounter. Since Klint is at Princeton and is very busy, I would not be able to meet with him very often, so most of our communication would be done via email.

2.2.2. Field Work Sites

Dwight Englewood School, Hajjar STEM Center
315 E Palisade Ave, Englewood, NJ 07631

Reasoning: Since this is a coding project, I will be able to work on it from anywhere I want, as long as there is WiFi. That means that I could work on my project during a free period. The Hajjar STEM center is the best place for me, as it is the one place in the school where I can be the most productive.

My House
178 Undercliff Ave, Edgewater, NJ 07020

Reasoning: The flexibility of my project also allows me to work from home. This is where I envision the bulk of my project taking place, as I am the most productive when I

am at my desk. I would also have access to a laptop with a GPU, which would allow me to gather data faster.

2.3. **Hours Spent Per Week**

Week Number	Date Range	Total Hours
Week # 1	2/5-2/11	11 Hrs. 35 min.
Week # 2	2/12-2/18	5 Hrs. 45 min.
Week # 3	2/19-2/25	7 Hrs
Week # 4	2/26-3/04	16 Hrs.
Week # 5	3/05-3/11	6 Hrs.
Week # 6	3/12-3/18	9 Hrs. 15 min.
Week # 7	3/19-3/25	8 Hrs.
Week # 8	3/26-4/01	6 Hrs. 45 min.
Week # 9	4/02-4/08	13 Hrs.
Week # 10	4/09-4/15	10 Hrs. 30 min.
Week # 11	4/16-4/22	16 Hrs. 30 min.
Week # 12	4/23-4/29	15 Hrs.
Week # 12	4/30-5/06	20 Hrs.

2.4. **Bibliography from Both Semesters**

[1] Abien Fred M. Agarap. Deep Learning using Rectified Linear Units (ReLU).

[2] Jimmy Ba and Diederik P. Kingma. Adam: A Method for Stochastic Optimization. 2014.

[3] Beom. CRNN (CNN+RNN) for OCR using Keras / License Plate Recognition. 2019.

[4] Yushi Chen, Xiao Liu, Bing Shuai, Bing Wang, Gang Wang, Xingxing Wang, and Zhen Zuo. Convolutional Recurrent Neural Networks: Learning Spatial Dependencies for Image Representation. *Computer Vision Foundation*, 2015.

[5] Dan Ciresan and Ueli Meier and Jürgen Schiduber. Multi-column deep neural networks for image classification. Technical report, Dalle Molle Institute for Artificial Intelligence, 2012.

[6] Trevor Darrell, Jeff Donahue, Sergio Guadarrama, Lisa Anne Hendricks, Marcus Rohrbach, Kate Saenko, and Subhashini Venugopalan. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. *Computer Vision Foundation*, 2014.

[7] Jacques-Louis David. The Death Of Socrates, 1787.

[8] Kate Fehlhaber. Hubel and Wiesel And The Neural Basis of Visual Perception. *Knowing Neurons*, 2014.

[9] Kunihiko Fukushima. Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 1980.

[10] Aurelien Geron. *Hands-On Machine LEarning wiht Scikit-Learn & TensorFlow*. O'Reilly, 2017.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Yanming Guo, Yu Liu, Erwin M. Bakker, Yuanhao Guo, and Michael S. Lew. CNN-RNN: a large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, 2018.

[13] Sepp Hochreiter and Jurgen Schmidhuber. Long Short Term Memory. *Neural Computation*, 1997.

[14] Jeremy Howard et al. Deep Learning for Coders, 2018. https://github.com/fastai/fastai.

[15] Chang Huang, Zhiheng Huang, Junhua Mao, Jiang Wang, and Yi Yang. CNN-RNN: A Unified Framework for Multi-label Image Classification. *Computer Vision Foundation*, 2016.

[16] Vijayabhaskar J. Multi-label image classification Tutorial with Keras ImageDataGenerator. 2019.

[17] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. *Andrej Karpathy Blog*.

[18] KeepLearning. InceptionV3 baseline. *Kaggle*, 2018. https://www.kaggle.com/mathormad/inceptionv3-baseline-lb-0-379.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

[20] Fred Lambert. Watch a Tesla Drive in Paris Through The Eyes of Autopilot. *Electrek*, 2018.
[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropogation Applied to Handwritten Zip Code Recogntion. *MIT*, 1989.
[22] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object Recognition with Gradient-Based Learning. 1998.
[23] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition 2016.
[24] Jianqiang Ma. All of Recurrent Neural Networks. *Medium*, 2016.
[25] Michael Nielson. *Neural Networks and Deep Learning*. Determination Press, 2015. https://neuralnetworksanddeeplearning.com/.
[26] nor. Keras InceptionResNetV2 (resize139x139). *Kaggle*, 2018. https://www.kaggle.com/wordroid/keras-inceptionresnetv2-resize139x139.
[27] Christopher Olah. Understanding LSTM Networks. *colah's Blog*, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[28] Barnabas Poczos. Convolutional Neural Networks.
[29] Vadim V. Romanuke. Appropriate Number of Standard 22 Max Pooling Layers and Their Allocation in Convolutional Neural Networks for Diverse and Heterogeneous Datasets . *De Gruyter*.
[30] Jurgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *The Swiss AI Lab IDSIA*, 2014.
[31] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. 2015.
[32] Devin P Sullivan, Casper F Winsnes, Lovisa Åkesson, Martin Hjelmare, Mikaela Wiking, Rutger Schutten, Linzi Campbell, Hjalti Leifsson, Scott Rhodes, Andie Nordgren, Kevin Smith, Bernard Revaz, Bergur Finnbogason, Attila Szantner, and Emma Lundberg. Deep Learning Is Combined With Massive-Scale Citizen Science To Improve Large-Scale Image Classification. *Nature Biotechnology*, 2018.
[33] Rob Verger. Facebook used billions of hashtagged Instagram photos to train its AI. *Popular Science*.
[34] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *CoRR*, 2015.
[35] Dong-Qing Zhang. Image Recognition Using Scale Recurrent Neural Networks. *ImaginationAI LLC*.
[36] Marrian Zhou. Amazon Go Expansion Continues With Store in Chicago. *C Net*, 2018.

3. Evidence

3.1. Evidence # 1: Code

Evidence # 1: Code

May 18, 2019

1 Multi Label Classification With MIML dataset

After all that had occured with the Protiens dataset, I could not figure out what was wrong. After doing some research I decided to follow this https://medium.com/@vijayabhaskar96/multi-label-image-classification-tutorial-with-keras-imagedatagenerator-cd541f8eaf24 but with my model instead of the one that the person builds

```
In [1]: import cv2
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import f1_score
        from sklearn.utils import shuffle
        import pandas as pd
        import numpy as np


        from PIL import Image
        #from tensorflow import keras

        import tensorflow as tf
        from imgaug import augmenters as iaa
        from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
        from tensorflow.keras.models import Sequential, Model
        from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,LSTM,Dense,Dropout
        from tensorflow.keras.layers import BatchNormalization,Activation, Flatten
        from tensorflow.keras.layers import Concatenate as con
        from tensorflow.keras.optimizers import Adam
        from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau,ModelCheckpoint
        from tensorflow.keras import backend as K

        import os

In [2]: print(os.getcwd())

/home/paperspace/fastai/focus


In [3]: df = pd.read_csv("./miml_dataset/miml_labels_1.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	Filenames	desert	mountains	sea	sunset	trees
0	1.jpg	1	0	0	0	0
1	2.jpg	1	0	0	0	0
2	3.jpg	1	0	0	0	0
3	4.jpg	1	1	0	0	0
4	5.jpg	1	0	0	0	0

In [5]:
```
columns = ["desert","mountains", "sea","sunset","trees"]

datagen = IDG(rescale = 1./255.)
test_datagen = IDG(rescale = 1./255.)

train_generator = datagen.flow_from_dataframe(
dataframe = df[:1800],
directory = "./miml_dataset/images",
x_col = "Filenames",
y_col = columns,
batch_size = 32,
seed = 42,
shuffle = True,
class_mode = "other",
target_size = (100,100))

valid_generator = datagen.flow_from_dataframe(
dataframe = df[1800:1900],
directory = "./miml_dataset/images",
x_col = "Filenames",
y_col = columns,
batch_size = 32,
seed = 42,
shuffle = True,
class_mode = "other",
target_size = (100,100))

test_generator = datagen.flow_from_dataframe(
dataframe = df[1900:],
directory = "./miml_dataset/images",
x_col = "Filenames",
y_col = columns,
batch_size = 1,
seed = 42,
shuffle = True,
class_mode = "other",
target_size = (100,100))
```

```
Found 1800 images.
Found 100 images.
```

```
Found 100 images.


In [6]: n_classes = 5

In [7]: input_shape=(100,100,3)

        inputs = Input(name='the_input', shape=input_shape, dtype='float32')

               # Convolution layer (VGG)
        inner = Conv2D(64, (3, 3), padding='same', name='conv1',
                       kernel_initializer='he_normal')(inputs)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)

        inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)

        inner = Conv2D(128, (3, 3), padding='same', name='conv3',
                       kernel_initializer='he_normal')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = Conv2D(128, (3, 3), padding='same', name='conv4',
                       kernel_initializer='he_normal')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)

        inner = Conv2D(256, (3, 3), padding='same', name='conv5',
                       kernel_initializer='he_normal')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = Conv2D(256, (3, 3), padding='same', name='conv6',
                       kernel_initializer='he_normal')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = Conv2D(256, (3, 3), padding='same', name='conv7',
                       kernel_initializer='he_normal')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = MaxPooling2D(pool_size=(1, 2), name='max3')(inner)

        inner = Conv2D(512, (3, 3), padding='same', name='conv8',
                       kernel_initializer='he_normal')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = Conv2D(512, (3, 3), padding='same',
                       kernel_initializer='he_normal' ,name='conv9')(inner)
        inner = BatchNormalization()(inner)
```

```
        inner = Activation('relu')(inner)
        inner = Conv2D(512, (2, 2), padding='same',
                       kernel_initializer='he_normal', name='con10')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = MaxPooling2D(pool_size=(1, 2), name='max4')(inner)

        inner = Conv2D(512, (2, 2), padding='same',
                       kernel_initializer='he_normal', name='con11')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
        inner = Conv2D(512, (2, 2), padding='same',
                       kernel_initializer='he_normal', name='con12')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)

        inner = Dense(64, activation='relu',
                      kernel_initializer='he_normal', name='dense_small')(inner)


        inner = Flatten()(inner)
        inner = Dense(4096, kernel_initializer='he_normal', name = 'fc1')(inner)
        inner = Activation('relu')(inner)
        inner = Dropout(0.3)(inner)
        inner = Dense(4096, kernel_initializer = 'he_normal',name = 'fc2')(inner)
        inner = Activation('relu')(inner)
        inner=Dropout(0.3)(inner)
        finalPred = Dense(5,kernel_initializer = 'he_normal',
                          name = 'dense_final', activation='sigmoid')(inner)
```

```
WARNING:tensorflow:From /home/paperspace/anaconda3/envs/fastai/lib/python3.7/site-packages/tenso
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/paperspace/anaconda3/envs/fastai/lib/python3.7/site-packages/tenso
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
In [8]: def f1(y_true, y_pred):
            tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
            fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
            fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

            p = tp / (tp + fp + K.epsilon())
            r = tp / (tp + fn + K.epsilon())

            f1 = 2*p*r / (p+r+K.epsilon())
            f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
            return K.mean(f1)
```

```
In [9]: cnn = Model(inputs=inputs, outputs= finalPred)

In [19]: cnn.compile(optimizer=Adam(decay=1e-6),
                     loss= "binary_crossentropy",
                       metrics=[f1,'accuracy'])

In [20]: cnn.summary()
```

Layer (type)	Output Shape	Param #
the_input (InputLayer)	(None, 100, 100, 3)	0
conv1 (Conv2D)	(None, 100, 100, 64)	1792
batch_normalization_v1 (Batc	(None, 100, 100, 64)	256
activation (Activation)	(None, 100, 100, 64)	0
max1 (MaxPooling2D)	(None, 50, 50, 64)	0
conv3 (Conv2D)	(None, 50, 50, 128)	73856
batch_normalization_v1_1 (Ba	(None, 50, 50, 128)	512
activation_1 (Activation)	(None, 50, 50, 128)	0
conv4 (Conv2D)	(None, 50, 50, 128)	147584
batch_normalization_v1_2 (Ba	(None, 50, 50, 128)	512
activation_2 (Activation)	(None, 50, 50, 128)	0
max2 (MaxPooling2D)	(None, 25, 25, 128)	0
conv5 (Conv2D)	(None, 25, 25, 256)	295168
batch_normalization_v1_3 (Ba	(None, 25, 25, 256)	1024
activation_3 (Activation)	(None, 25, 25, 256)	0
conv6 (Conv2D)	(None, 25, 25, 256)	590080
batch_normalization_v1_4 (Ba	(None, 25, 25, 256)	1024
activation_4 (Activation)	(None, 25, 25, 256)	0
conv7 (Conv2D)	(None, 25, 25, 256)	590080

```
_________________________________________________________________
batch_normalization_v1_5 (Ba (None, 25, 25, 256)       1024
_________________________________________________________________
activation_5 (Activation)    (None, 25, 25, 256)       0
_________________________________________________________________
max3 (MaxPooling2D)          (None, 25, 12, 256)       0
_________________________________________________________________
conv8 (Conv2D)               (None, 25, 12, 512)       1180160
_________________________________________________________________
batch_normalization_v1_6 (Ba (None, 25, 12, 512)       2048
_________________________________________________________________
activation_6 (Activation)    (None, 25, 12, 512)       0
_________________________________________________________________
conv9 (Conv2D)               (None, 25, 12, 512)       2359808
_________________________________________________________________
batch_normalization_v1_7 (Ba (None, 25, 12, 512)       2048
_________________________________________________________________
activation_7 (Activation)    (None, 25, 12, 512)       0
_________________________________________________________________
con10 (Conv2D)               (None, 25, 12, 512)       1049088
_________________________________________________________________
batch_normalization_v1_8 (Ba (None, 25, 12, 512)       2048
_________________________________________________________________
activation_8 (Activation)    (None, 25, 12, 512)       0
_________________________________________________________________
max4 (MaxPooling2D)          (None, 25, 6, 512)        0
_________________________________________________________________
con11 (Conv2D)               (None, 25, 6, 512)        1049088
_________________________________________________________________
batch_normalization_v1_9 (Ba (None, 25, 6, 512)        2048
_________________________________________________________________
activation_9 (Activation)    (None, 25, 6, 512)        0
_________________________________________________________________
con12 (Conv2D)               (None, 25, 6, 512)        1049088
_________________________________________________________________
batch_normalization_v1_10 (B (None, 25, 6, 512)        2048
_________________________________________________________________
activation_10 (Activation)   (None, 25, 6, 512)        0
_________________________________________________________________
dense_small (Dense)          (None, 25, 6, 64)         32832
_________________________________________________________________
flatten (Flatten)            (None, 9600)              0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              39325696
_________________________________________________________________
activation_11 (Activation)   (None, 4096)              0
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
```

```
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
activation_12 (Activation)   (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_final (Dense)          (None, 5)                 20485
=================================================================
Total params: 64,560,709
Trainable params: 64,553,413
Non-trainable params: 7,296
_________________________________________________________________
```

```
In [21]: input_shape=(100,100,3)

         inputz = Input(name='the_input', shape=input_shape, dtype='float32')

         inner = Conv2D(64, (3, 3), padding='same', name='conv1',
                        kernel_initializer='he_normal')(inputz)
         inner = BatchNormalization()(inner)
         inner = Activation('relu')(inner)
         inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)

         inner = Conv2D(128, (3, 3), padding='same', name='conv2',
                        kernel_initializer='he_normal')(inner)
         inner = BatchNormalization()(inner)
         inner = Activation('relu')(inner)
         inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)

         inner = Conv2D(256, (3, 3), padding='same', name='conv3',
                        kernel_initializer='he_normal')(inner)
         inner = BatchNormalization()(inner)
         inner = Activation('relu')(inner)
         inner = Conv2D(256, (3, 3), padding='same', name='conv4',
                        kernel_initializer='he_normal')(inner)
         inner = BatchNormalization()(inner)
         inner = Activation('relu')(inner)
         inner = MaxPooling2D(pool_size=(1, 2), name='max3')(inner)

         inner = Conv2D(512, (3, 3), padding='same', name='conv5',
                        kernel_initializer='he_normal')(inner)
         inner = BatchNormalization()(inner)
         inner = Activation('relu')(inner)
         inner = Conv2D(512, (3, 3), padding='same', name='conv6')(inner)
         inner = BatchNormalization()(inner)
         inner = Activation('relu')(inner)
```

```
        inner = MaxPooling2D(pool_size=(1, 2), name='max4')(inner)
        inner = Conv2D(512, (2, 2), padding='same',
                       kernel_initializer='he_normal', name='con7')(inner)
        inner = BatchNormalization()(inner)
        inner = Activation('relu')(inner)
            # CNN to RNN
        inner = tf.keras.layers.Reshape(target_shape=((25, 512*6)), name='reshape')(inner)
        inner = Dense(64, activation='relu', kernel_initializer='he_normal',
                      name='dense_smaller')(inner)


            # RNN layer
        lstm_1 = LSTM(256, return_sequences=False,
                      kernel_initializer='he_normal', name='lstm1')(inner)
        lstm_1b = LSTM(256, return_sequences=False, go_backwards=True,
                       kernel_initializer='he_normal', name='lstm1_b')(inner)
        reversed_lstm_1b = tf.keras.layers.Lambda(lambda inputTensor:
                                                  K.reverse(inputTensor, axes=1)) (lstm_1b)

        lstm1_merged = tf.keras.layers.add([lstm_1, reversed_lstm_1b])
        lstm1_merged = BatchNormalization()(lstm_1)


        finalPred1 = Dense(n_classes, kernel_initializer = 'he_normal',name='dense',
                           activation = 'sigmoid')(lstm1_merged)
```

```
In [22]: cnn_lstm = Model(inputs=inputz, outputs= finalPred1 )
```

```
In [23]: cnn_lstm.compile(optimizer=Adam(lr=0.0001,  decay=1e-6), loss= "binary_crossentropy",
                        metrics=[ 'accuracy' ,f1])
```

```
In [24]: cnn_lstm.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, 100, 100, 3)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 100, 100, 64)      1792
_________________________________________________________________
batch_normalization_v1_19 (B (None, 100, 100, 64)      256
_________________________________________________________________
activation_20 (Activation)   (None, 100, 100, 64)      0
_________________________________________________________________
max1 (MaxPooling2D)          (None, 50, 50, 64)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 50, 50, 128)       73856
_________________________________________________________________
batch_normalization_v1_20 (B (None, 50, 50, 128)       512
```

```
_________________________________________________________________
activation_21 (Activation)   (None, 50, 50, 128)       0
_________________________________________________________________
max2 (MaxPooling2D)          (None, 25, 25, 128)       0
_________________________________________________________________
conv3 (Conv2D)               (None, 25, 25, 256)       295168
_________________________________________________________________
batch_normalization_v1_21 (B (None, 25, 25, 256)       1024
_________________________________________________________________
activation_22 (Activation)   (None, 25, 25, 256)       0
_________________________________________________________________
conv4 (Conv2D)               (None, 25, 25, 256)       590080
_________________________________________________________________
batch_normalization_v1_22 (B (None, 25, 25, 256)       1024
_________________________________________________________________
activation_23 (Activation)   (None, 25, 25, 256)       0
_________________________________________________________________
max3 (MaxPooling2D)          (None, 25, 12, 256)       0
_________________________________________________________________
conv5 (Conv2D)               (None, 25, 12, 512)       1180160
_________________________________________________________________
batch_normalization_v1_23 (B (None, 25, 12, 512)       2048
_________________________________________________________________
activation_24 (Activation)   (None, 25, 12, 512)       0
_________________________________________________________________
conv6 (Conv2D)               (None, 25, 12, 512)       2359808
_________________________________________________________________
batch_normalization_v1_24 (B (None, 25, 12, 512)       2048
_________________________________________________________________
activation_25 (Activation)   (None, 25, 12, 512)       0
_________________________________________________________________
max4 (MaxPooling2D)          (None, 25, 6, 512)        0
_________________________________________________________________
con7 (Conv2D)                (None, 25, 6, 512)        1049088
_________________________________________________________________
batch_normalization_v1_25 (B (None, 25, 6, 512)        2048
_________________________________________________________________
activation_26 (Activation)   (None, 25, 6, 512)        0
_________________________________________________________________
reshape (Reshape)            (None, 25, 3072)          0
_________________________________________________________________
dense_smaller (Dense)        (None, 25, 64)            196672
_________________________________________________________________
lstm1 (LSTM)                 (None, 256)               328704
_________________________________________________________________
batch_normalization_v1_26 (B (None, 256)               1024
_________________________________________________________________
dense (Dense)                (None, 5)                 1285
```

```
=================================================================
Total params: 6,086,597
Trainable params: 6,081,605
Non-trainable params: 4,992
_________________________________________________________________
```

```
In [25]: cnn.reset_states()
```

```
In [26]: def generator_wrapper(generator):
             for batch_x,batch_y in generator:
                 yield (batch_x,[batch_y[:,i] for i in range(5)])
```

```
In [27]: STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
         STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
         STEP_SIZE_TEST=100//test_generator.batch_size


         filepath = "CNN_MIML-{epoch:02d}.hdf5"

         checkpoint = ModelCheckpoint(filepath, monitor='val_f1', verbose=1,
                                      save_best_only=True, mode='max')

         callback =  ReduceLROnPlateau(monitor='val_f1', factor=0.5, patience=3,
                               min_delta=0.005, mode='max', cooldown=3, verbose=1)

         history = cnn.fit_generator(generator=train_generator,
                             steps_per_epoch=STEP_SIZE_TRAIN,
                             validation_data=valid_generator,
                             validation_steps=STEP_SIZE_VALID,
                             epochs=15,
                             callbacks = [callback, checkpoint]
         )
```

```
Epoch 1/15
4/4 [==============================] - 4s 1s/step - loss: 0.4050 - f1: 0.1592 - acc: 0.8660

Epoch 00001: val_f1 improved from -inf to 0.15924, saving model to CNN_MIML-01.hdf5
57/57 [==============================] - 349s 6s/step - loss: 0.8118 - f1: 0.2604 - acc: 0.7357
Epoch 2/15
4/4 [==============================] - 4s 922ms/step - loss: 0.7642 - f1: 0.0704 - acc: 0.6400

Epoch 00002: val_f1 did not improve from 0.15924
57/57 [==============================] - 335s 6s/step - loss: 0.4776 - f1: 0.3806 - acc: 0.7833
Epoch 3/15
4/4 [==============================] - 3s 802ms/step - loss: 0.7224 - f1: 0.0847 - acc: 0.6260

Epoch 00003: val_f1 did not improve from 0.15924
57/57 [==============================] - 304s 5s/step - loss: 0.4257 - f1: 0.4554 - acc: 0.8106
```

```
Epoch 4/15
4/4 [==============================] - 3s 788ms/step - loss: 1.0864 - f1: 0.0310 - acc: 0.6160

Epoch 00004: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 00004: val_f1 did not improve from 0.15924
57/57 [==============================] - 292s 5s/step - loss: 0.4134 - f1: 0.4757 - acc: 0.8132
Epoch 5/15
4/4 [==============================] - 3s 719ms/step - loss: 0.7727 - f1: 0.0973 - acc: 0.6240

Epoch 00005: val_f1 did not improve from 0.15924
57/57 [==============================] - 292s 5s/step - loss: 0.3856 - f1: 0.5128 - acc: 0.8301
Epoch 6/15
4/4 [==============================] - 3s 781ms/step - loss: 0.7187 - f1: 0.0813 - acc: 0.6820

Epoch 00006: val_f1 did not improve from 0.15924
57/57 [==============================] - 291s 5s/step - loss: 0.3621 - f1: 0.5433 - acc: 0.8448
Epoch 7/15
4/4 [==============================] - 3s 729ms/step - loss: 0.6289 - f1: 0.1183 - acc: 0.7620

Epoch 00007: val_f1 did not improve from 0.15924
57/57 [==============================] - 292s 5s/step - loss: 0.3490 - f1: 0.5592 - acc: 0.8501
Epoch 8/15
4/4 [==============================] - 3s 729ms/step - loss: 1.0393 - f1: 0.0480 - acc: 0.6060

Epoch 00008: val_f1 did not improve from 0.15924
57/57 [==============================] - 292s 5s/step - loss: 0.3488 - f1: 0.5600 - acc: 0.8496
Epoch 9/15
4/4 [==============================] - 3s 741ms/step - loss: 0.4482 - f1: 0.1625 - acc: 0.8400

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 00009: val_f1 improved from 0.15924 to 0.16249, saving model to CNN_MIML-09.hdf5
57/57 [==============================] - 293s 5s/step - loss: 0.3308 - f1: 0.5890 - acc: 0.8557
Epoch 10/15
4/4 [==============================] - 3s 789ms/step - loss: 0.3214 - f1: 0.1889 - acc: 0.8440

Epoch 00010: val_f1 improved from 0.16249 to 0.18890, saving model to CNN_MIML-10.hdf5
57/57 [==============================] - 294s 5s/step - loss: 0.3030 - f1: 0.6104 - acc: 0.8706
Epoch 11/15
4/4 [==============================] - 3s 805ms/step - loss: 0.3352 - f1: 0.1922 - acc: 0.8760

Epoch 00011: val_f1 improved from 0.18890 to 0.19220, saving model to CNN_MIML-11.hdf5
57/57 [==============================] - 295s 5s/step - loss: 0.2937 - f1: 0.6365 - acc: 0.8750
Epoch 12/15
4/4 [==============================] - 3s 777ms/step - loss: 0.3078 - f1: 0.2008 - acc: 0.8360

Epoch 00012: val_f1 improved from 0.19220 to 0.20083, saving model to CNN_MIML-12.hdf5
```

```
57/57 [==============================] - 294s 5s/step - loss: 0.2828 - f1: 0.6352 - acc: 0.8796
Epoch 13/15
4/4 [==============================] - 3s 848ms/step - loss: 0.3374 - f1: 0.1941 - acc: 0.8680

Epoch 00013: val_f1 did not improve from 0.20083
57/57 [==============================] - 293s 5s/step - loss: 0.2811 - f1: 0.6439 - acc: 0.8814
Epoch 14/15
4/4 [==============================] - 3s 754ms/step - loss: 0.3326 - f1: 0.2214 - acc: 0.8680

Epoch 00014: val_f1 improved from 0.20083 to 0.22137, saving model to CNN_MIML-14.hdf5
57/57 [==============================] - 295s 5s/step - loss: 0.2634 - f1: 0.6712 - acc: 0.8884
Epoch 15/15
4/4 [==============================] - 3s 798ms/step - loss: 0.2626 - f1: 0.2138 - acc: 0.8880

Epoch 00015: val_f1 did not improve from 0.22137
57/57 [==============================] - 293s 5s/step - loss: 0.2530 - f1: 0.6881 - acc: 0.8980


In [29]: cnn_lstm.reset_states

Out[29]: <bound method Network.reset_states of <tensorflow.python.keras.engine.training.Model ob

In [30]: STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
         STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
         STEP_SIZE_TEST=100//test_generator.batch_size
         filepath = "CNNLSTM-miml-{epoch:02d}.hdf5"


         checkpoint1 = ModelCheckpoint(filepath, monitor='loss', verbose=1,
                                       save_best_only=True, mode='max')
         hist = cnn_lstm.fit_generator(generator=train_generator,
                             steps_per_epoch=STEP_SIZE_TRAIN,
                             validation_data=valid_generator,
                             validation_steps=STEP_SIZE_VALID,
                             epochs=15,
                             callbacks = [callback,checkpoint1]
         )

Epoch 1/15
4/4 [==============================] - 2s 581ms/step - loss: 0.7472 - acc: 0.3500 - f1: 0.1465

Epoch 00001: loss improved from -inf to 0.62902, saving model to CNNLSTM-miml-01.hdf5
57/57 [==============================] - 184s 3s/step - loss: 0.6326 - acc: 0.6774 - f1: 0.4456
Epoch 2/15
4/4 [==============================] - 2s 526ms/step - loss: 0.8066 - acc: 0.4000 - f1: 0.1458

Epoch 00002: loss did not improve from 0.62902
57/57 [==============================] - 182s 3s/step - loss: 0.4934 - acc: 0.7772 - f1: 0.5186
Epoch 3/15
```

```
4/4 [==============================] - 2s 530ms/step - loss: 0.7933 - acc: 0.4840 - f1: 0.1219

Epoch 00003: loss did not improve from 0.62902
57/57 [==============================] - 181s 3s/step - loss: 0.4106 - acc: 0.8330 - f1: 0.5778
Epoch 4/15
4/4 [==============================] - 2s 508ms/step - loss: 0.8910 - acc: 0.4780 - f1: 0.0984

Epoch 00004: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.

Epoch 00004: loss did not improve from 0.62902
57/57 [==============================] - 181s 3s/step - loss: 0.3532 - acc: 0.8677 - f1: 0.6177
Epoch 5/15
4/4 [==============================] - 2s 456ms/step - loss: 0.6324 - acc: 0.6400 - f1: 0.1632

Epoch 00005: loss did not improve from 0.62902
57/57 [==============================] - 179s 3s/step - loss: 0.2866 - acc: 0.9047 - f1: 0.6643
Epoch 6/15
4/4 [==============================] - 2s 490ms/step - loss: 0.5851 - acc: 0.7520 - f1: 0.1741

Epoch 00006: loss did not improve from 0.62902
57/57 [==============================] - 177s 3s/step - loss: 0.2221 - acc: 0.9424 - f1: 0.7151
Epoch 7/15
4/4 [==============================] - 2s 500ms/step - loss: 0.5375 - acc: 0.7920 - f1: 0.1838

Epoch 00007: loss did not improve from 0.62902
57/57 [==============================] - 179s 3s/step - loss: 0.1984 - acc: 0.9474 - f1: 0.7368
Epoch 8/15
4/4 [==============================] - 2s 515ms/step - loss: 0.4715 - acc: 0.8200 - f1: 0.2027

Epoch 00008: loss did not improve from 0.62902
57/57 [==============================] - 177s 3s/step - loss: 0.1682 - acc: 0.9609 - f1: 0.7663
Epoch 9/15
4/4 [==============================] - 2s 531ms/step - loss: 0.5542 - acc: 0.7920 - f1: 0.1954

Epoch 00009: loss did not improve from 0.62902
57/57 [==============================] - 179s 3s/step - loss: 0.1478 - acc: 0.9666 - f1: 0.7838
Epoch 10/15
4/4 [==============================] - 2s 496ms/step - loss: 0.4279 - acc: 0.8100 - f1: 0.2396

Epoch 00010: loss did not improve from 0.62902
57/57 [==============================] - 177s 3s/step - loss: 0.1273 - acc: 0.9724 - f1: 0.8058
Epoch 11/15
4/4 [==============================] - 2s 472ms/step - loss: 0.4004 - acc: 0.8520 - f1: 0.2359

Epoch 00011: loss did not improve from 0.62902
57/57 [==============================] - 179s 3s/step - loss: 0.1095 - acc: 0.9792 - f1: 0.8231
Epoch 12/15
4/4 [==============================] - 2s 518ms/step - loss: 0.3915 - acc: 0.8720 - f1: 0.2344
```

```
Epoch 00012: loss did not improve from 0.62902
57/57 [==============================] - 178s 3s/step - loss: 0.1062 - acc: 0.9806 - f1: 0.8278
Epoch 13/15
4/4 [==============================] - 2s 508ms/step - loss: 0.3235 - acc: 0.8700 - f1: 0.2527

Epoch 00013: loss did not improve from 0.62902
57/57 [==============================] - 178s 3s/step - loss: 0.0947 - acc: 0.9820 - f1: 0.8433
Epoch 14/15
4/4 [==============================] - 2s 468ms/step - loss: 0.5365 - acc: 0.8700 - f1: 0.2221

Epoch 00014: loss did not improve from 0.62902
57/57 [==============================] - 178s 3s/step - loss: 0.0849 - acc: 0.9872 - f1: 0.8577
Epoch 15/15
4/4 [==============================] - 2s 521ms/step - loss: 0.4106 - acc: 0.8700 - f1: 0.2333

Epoch 00015: loss did not improve from 0.62902
57/57 [==============================] - 179s 3s/step - loss: 0.0774 - acc: 0.9878 - f1: 0.8666
```

```
In [32]: test_generator.reset()
         cnn_preds = cnn.predict_generator(test_generator, steps = STEP_SIZE_TEST, verbose=1)
         cnnLSTM_preds = cnn_lstm.predict_generator(test_generator, steps = STEP_SIZE_TEST, verb

100/100 [==============================] - 8s 76ms/step
100/100 [==============================] - 4s 44ms/step
```

```
In [41]: predictions=[]
         for i in range(len(cnn_preds)):
             l = []
             for j in range(len(cnn_preds[i])):
                 if cnnLSTM_preds[i][j] > 0.5:
                     l.append(1)
                 else:
                     l.append(0)
             predictions.append(l)
         columns=["desert", "mountains", "sea", "sunset", "trees"]
         #columms should be the same order of y_col
         results=pd.DataFrame(predictions, columns=columns)
         results["Filenames"]=test_generator.filenames
         ordered_cols=["Filenames"]+columns
         results=results[ordered_cols]#To get the same column order
         print(results)
         results.to_csv("CNN_Pt2_results.csv",index=False)

   Filenames  desert  mountains  sea  sunset  trees
0   1901.jpg       0          0    0       1      1
1   1902.jpg       0          1    0       0      0
```

```
2   1903.jpg       0          0    0       0      1
3   1904.jpg       0          0    0       0      1
4   1905.jpg       0          0    1       0      0
5   1906.jpg       0          0    0       0      1
6   1907.jpg       1          0    0       0      1
7   1908.jpg       0          0    0       0      1
8   1909.jpg       0          1    0       0      1
9   1910.jpg       0          1    0       0      1
10  1911.jpg       0          0    0       0      1
11  1912.jpg       1          0    0       0      1
12  1913.jpg       1          0    1       0      1
13  1914.jpg       0          0    0       0      1
14  1915.jpg       0          1    0       0      1
15  1916.jpg       1          0    0       0      1
..       ...     ...        ...  ...     ...    ...

89  1990.jpg       0          1    1       0      0
90  1991.jpg       0          0    0       0      1
91  1992.jpg       0          0    0       0      1
92  1993.jpg       1          0    0       0      1
93  1994.jpg       0          1    0       0      1
94  1995.jpg       1          1    0       0      0
95  1996.jpg       0          0    0       0      1
96  1997.jpg       1          1    0       0      1
97  1998.jpg       0          1    0       0      1
98  1999.jpg       0          1    0       0      0
99  2000.jpg       0          0    0       0      1

[100 rows x 6 columns]
```

```
In [40]: predictions=[]
         for i in range(len(cnnLSTM_preds)):
             l = []
             for j in range(len(cnnLSTM_preds[i])):
                 if cnnLSTM_preds[i][j] > 0.5:
                     l.append(1)
                 else:
                     l.append(0)
             predictions.append(l)
         columns=["desert", "mountains", "sea", "sunset", "trees"]
         #columns should be the same order of y_col
         results=pd.DataFrame(predictions, columns=columns)
         results["Filenames"]=test_generator.filenames
         ordered_cols=["Filenames"]+columns
         results=results[ordered_cols]#To get the same column order
         print(results)
         results.to_csv("CNNLSTM_Pt2_results.csv",index=False)
```

	Filenames	desert	mountains	sea	sunset	trees
0	1901.jpg	0	0	0	1	1
1	1902.jpg	0	1	0	0	0
2	1903.jpg	0	0	0	0	1
3	1904.jpg	0	0	0	0	1
4	1905.jpg	0	0	1	0	0
5	1906.jpg	0	0	0	0	1
6	1907.jpg	1	0	0	0	1
7	1908.jpg	0	0	0	0	1
8	1909.jpg	0	1	0	0	1
9	1910.jpg	0	1	0	0	1
10	1911.jpg	0	0	0	0	1
11	1912.jpg	1	0	0	0	1
12	1913.jpg	1	0	1	0	1
13	1914.jpg	0	0	0	0	1
14	1915.jpg	0	1	0	0	1
15	1916.jpg	1	0	0	0	1
..	...	...	...	...	...	...
89	1990.jpg	0	1	1	0	0
90	1991.jpg	0	0	0	0	1
91	1992.jpg	0	0	0	0	1
92	1993.jpg	1	0	0	0	1
93	1994.jpg	0	1	0	0	1
94	1995.jpg	1	1	0	0	0
95	1996.jpg	0	0	0	0	1
96	1997.jpg	1	1	0	0	1
97	1998.jpg	0	1	0	0	1
98	1999.jpg	0	1	0	0	0
99	2000.jpg	0	0	0	0	1

[100 rows x 6 columns]

3.2. **Rationale for Evidence # 1**

It is only through the code that I could implement these networks. In order to implement these networks, I chose to use Python as the language and Keras as the main machine learning library. I did all of my work in Jupyter Notebook, so I was able to get my code like this. I chose this particular notebook as it shows my work with a multilabel dataset, and has both models in the file. This particular notebook shows all of the work that I did with the MIML dataset.

In this notebook, the first thing that happens is that I read all of the information in the CSV file and save into a variable named df (short for data frame). I then look at df to see if the `read_csv` function was successful. After that I create my three generators: `train_generator`, `valid_generator`, and `test_generator`. These generators are the only ways that I could get the image from my laptop to the actual model. I use the `flow_from_dataframe` in order to make it easier on myself. I split the entire data set by setting aside 80 % of the images for training, 10% for validating and 10% for testing. This split allows me to keep the majority for training and have enough to make accurate measurements at the end. After running this, the output shows that it was successful as it found all of the images.

After creating these generators I define both my CNN and CNNLSTM. For CNN, I used a VGG11, as it did not have too many parameters compared to the other VGG models. The VGG models are made up of many repetitive combinations of layers. This combination consists of different numbers of convolutional layers, followed by a max Pooling layer. As the code shows the number of filters and layers increase as you go down. Between these layers, I added some BatchNormalization, so that the training time is decreased and to improve

the overall performance of the network. Each layer takes the previous layer as its input. After that, all of the information gathered from the convolutional layers are refined by 4 Dense layers. The 64 neuron layer is meant to reduce the number of parameters, the two 4096 neuron layers are required by the network and the last 5 neuron layer is meant for the final decision. I use the relu activation function for the first three layers as it is the best for refining the information and a sigmoid activation function for the last layer as it is the best activation function to make the final decision for a multi-label dataset. In order for the network to also perform better, I added dropout between the layers in order to reduce the chances of the network focusing on one class rather than all of the classes. After that, I define the CNNLSTM by using the same CNN base, but I swap out the middle dense layers for a Bidirectional LSTM. A bidirectional LSTM allows me to see everything that the CNN has learned and even make connections between patterns that it saw earlier on in the network and patterns that it saw later on in the network. The bidirectional LSTM also allows the LSTM to process more information. To make a bidirectional LSTM, I trained two LSTMs on the same thing but reversed one of them. I then added the two together to combine both of their information. After the bidirectional LSTM, I put the final dense layer in order to get the classifications. The final step in making both networks is to compile them. I compiled both of them with the Adam optimizer and the `binary_crossentropy` loss function. For the optimizer, I set the decay to be 0.000001 as it decreases the learning rate, better improving the model. Since the learning rate starts off pretty high, I wanted to slowly decrease it over time. In order to do this, I used the decay in order to incrementally decrease it with each step. I used the Adam optimizer as it is currently the most popular optimizer as it achieves a better learning rate faster compared to the other optimizers. The `binary_crossentropy` function is used in order to accurately compute how much information the networks are

missing. It is best used for multilabel classification tasks, as it compiles the loss for each class separately and then combines all of them.

Once both networks were compiled, it was time to train and test them. In order to train both models, I used the `fit_generator` method as it handles the generators that I created before, the best. For the `steps_per_epoch` I put in the total number of images in the training set divided by the batch size and for the `validation_steps` I used the number of images in the test set divided by the batch size. These numbers are what are conventionally used as it allows the function to go over each of the images in the training and testing set faster. I made the epochs set to 15, as I felt that it would allow the network to learn as much as it could before it hit a wall. During training, I added two callbacks in order to make sure that my model did better and to save my progress. The model checkpoint allows me to save the model when the valid f1 score improves so that if something interrupts training, I can still have some data to work with. The ReduceLROnPlateau callback decreases the learning rate when the valid f1 score does not improve after three epochs, to help the model improve. With all of these values, I let the model both models train over the course of a few days. Once both models finished training it was time to test both models. I used the `predict_generator` method in order to gather all of the predictions. With both the models' predictions, I then outputted them into CSV files. In order to do this, I went through what the model outputted for each of the images, and if it was greater than 0.5, I counted it as a classification. Going through each of the classifications, I was finally left with the data I needed in order to analyze both networks.

3.3. **Evidence # 2: Training Logs**

3.3.1. **MNIST**

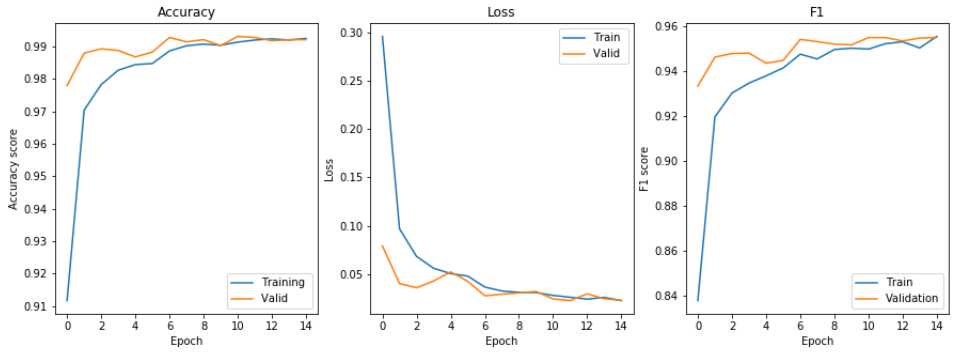

FIGURE 1. CNN

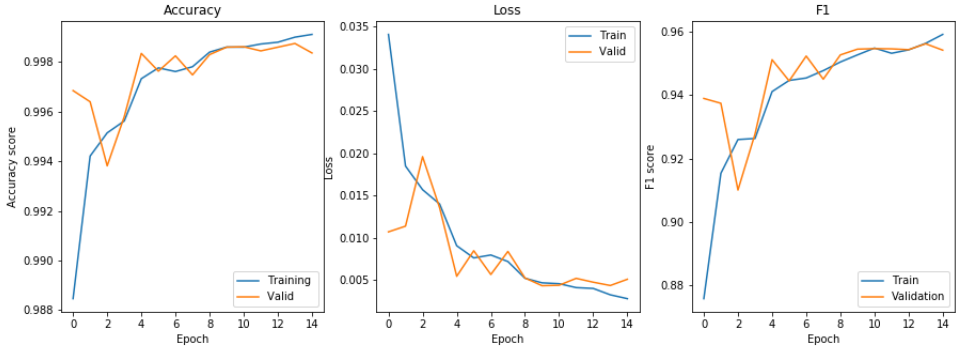

FIGURE 2. CNNLSTM

3.3.2. **FMNIST**

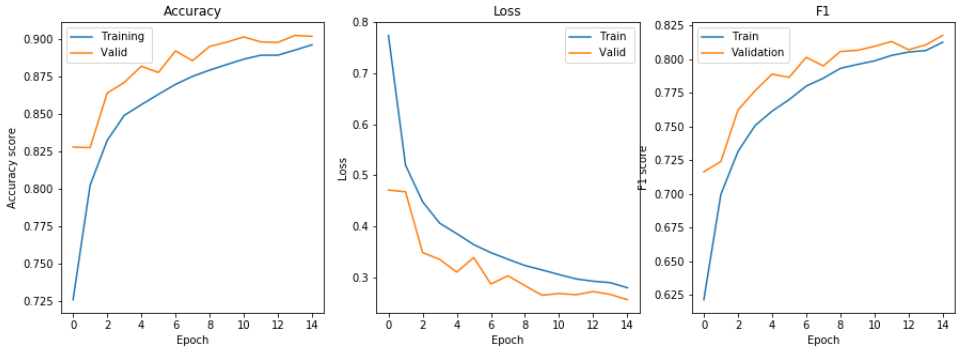

FIGURE 3. CNN

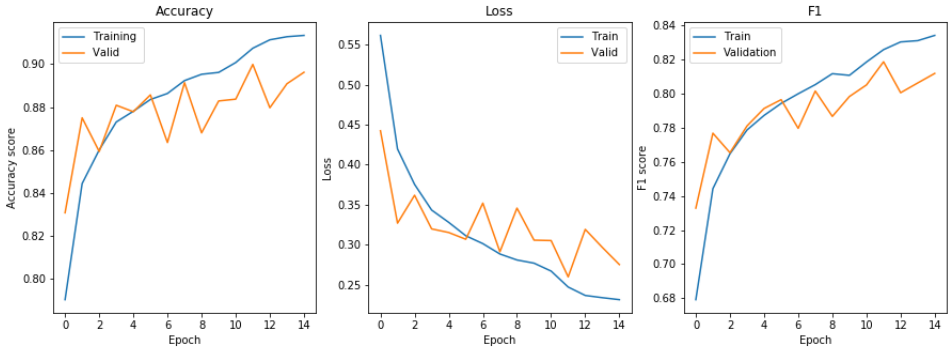

FIGURE 4. CNNLSTM

3.3.3. **MIML**

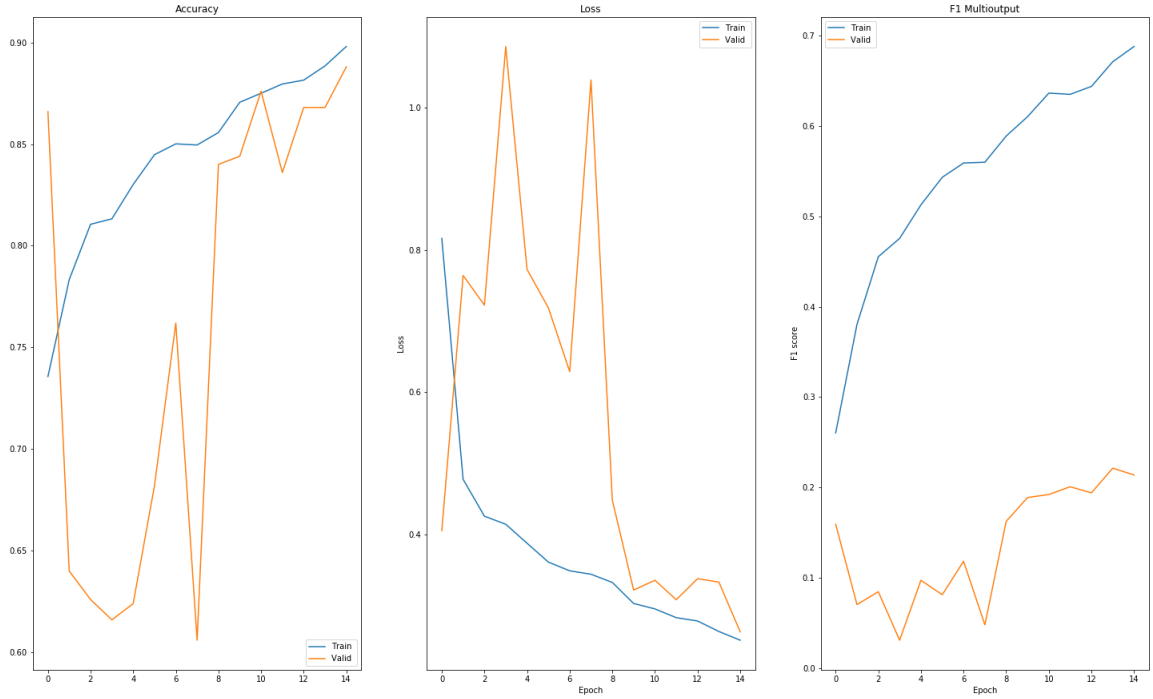

FIGURE 5. CNN

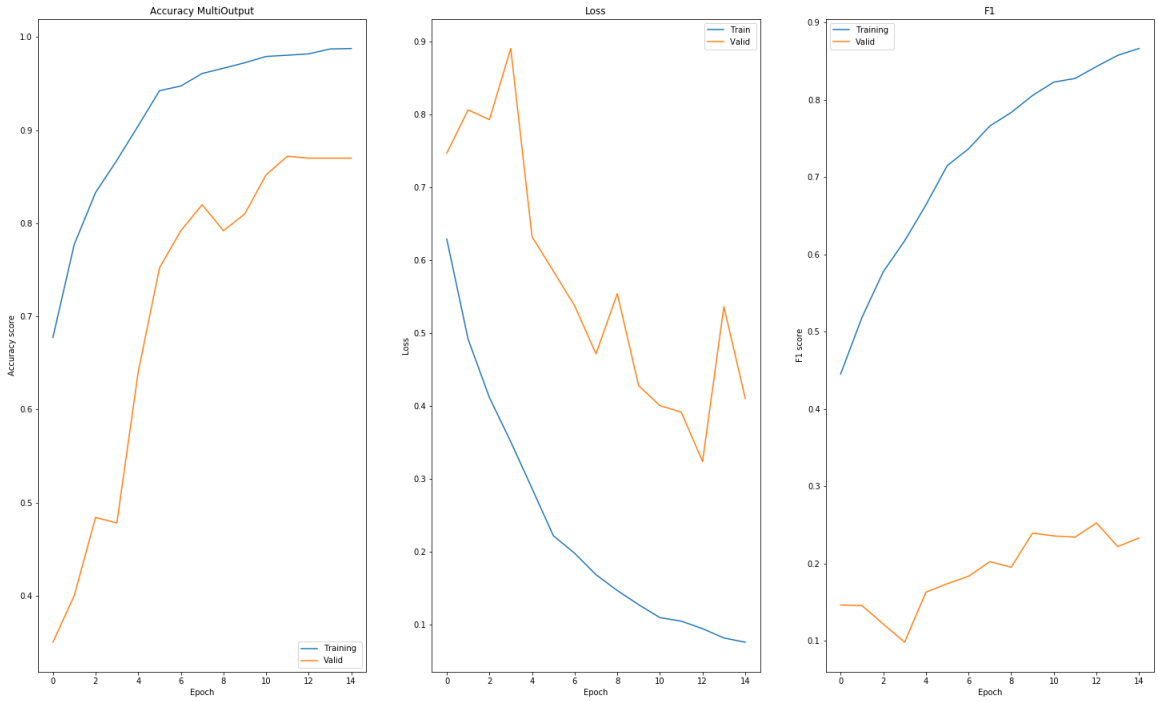

FIGURE 6. CNNLSTM

3.3.4. **Human Protein Atlas**

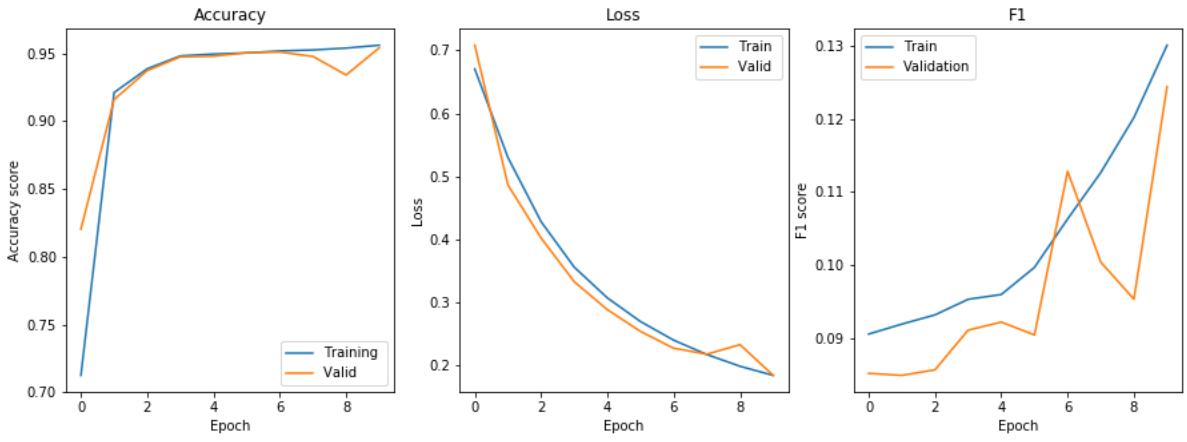

FIGURE 7. CNN

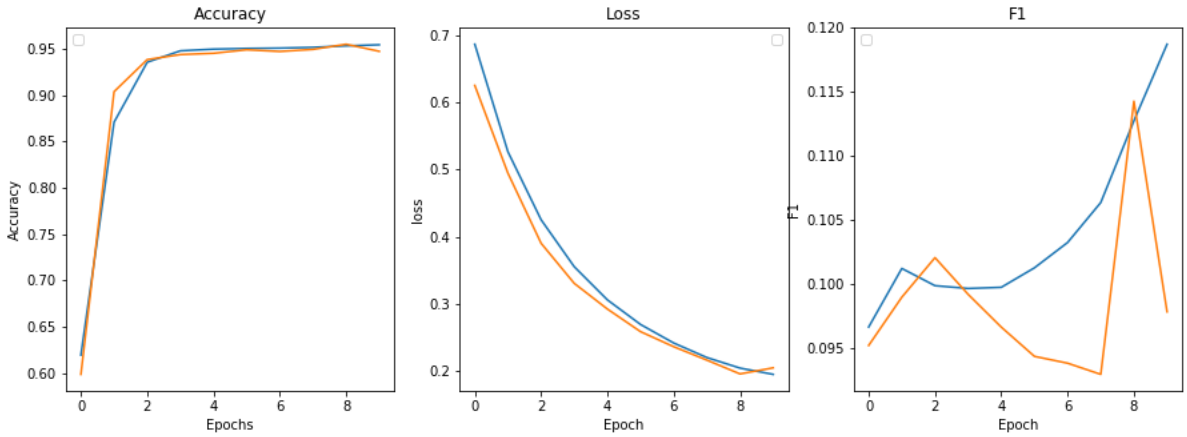

FIGURE 8. CNNLSTM

3.4. **Rationale for Evidence #2 Logs**

In order to get a neural network to learn how to classify images, it needs to be trained. To train a neural network, you feed it images and labels and the network will try to see the paterns between the images with the same labels. Training a neural network takes a long time, so it is very important to keep track of how the model is doing through out the entire training process. These logs are meant to show how the CNN and CNNLSTM does over the entire training process. With these logs, I am able to make a better to assess these models better and answer the questions *How does each model do on a multi-class dataset?*, *How does one assess these networks?*, and *How does each model do on a multi-labeled dataset?*

In order to assess the networks, you need to look at three different aspects of each graph: the metric, the scale, and the trend. For all of the datasets, the loss, accuracy and f1 score was tracked through out the training process. The loss tells us how much the network is prone to make bad classifications. A high loss means that the weights in the network are not optimized. Because of this fact, a low loss is desired and during training, the loss should be decreasing. Another metric used is the accuracy score. The accuracy is calculated by

dividing the number of correct classifications by the total number of images classified. A higher accuracy score is always desired and the overall trend should be increasing. The last metric that is the F1 score. The F1 score is similar to the accuracy score but allows you to see how accurate a model is when there is an uneven amount of classes. For example, if there were too many 9s in the MNIST dataset and not a lot 3s, the accuracy score may be high, but the f1 score may be a bit lower. F1 does act similarly to accuracy score, so you would want a similar trend to occur with that metric as well. Now looking at all of the graphs, no matter what the dataset was, both the CNN and CNNLSTM performed well. However, what really differentiated the CNNLSTM from the CNN was the scale on the right. With each graph, the CNNLSTM had a higher scale for accuracy and f1 and a lower scale for the losses. These values mean that even though both the CNN and CNNLSTM both improved over time, the CNNLSTM was performing better with each pass.

3.5. **Evidence # 3: Sample Classifications**

3.5.1. **MNIST**

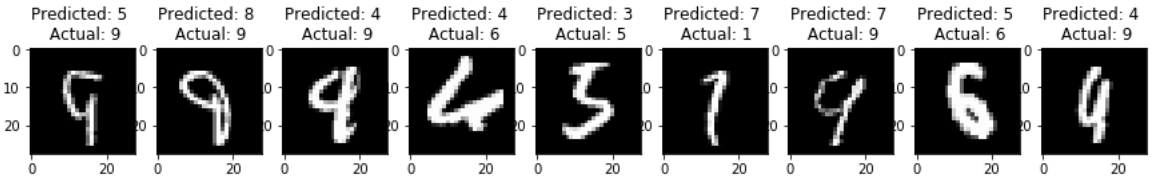

FIGURE 9. Some of the images that the CNN mislabeled.

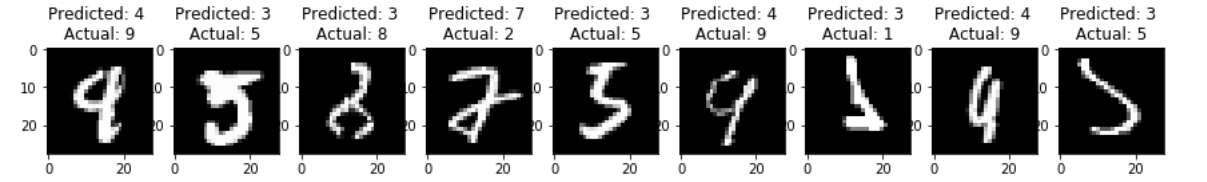

FIGURE 10. Some of the images that the CNNLSTM mislabeled.

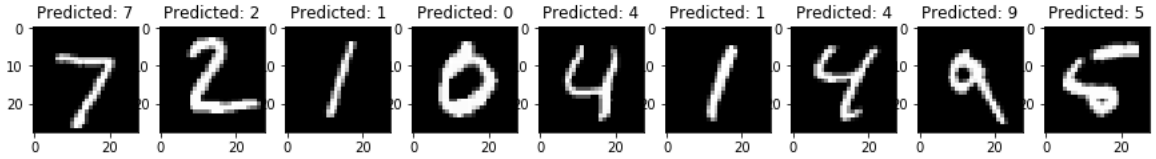

FIGURE 11. CORRECT BOTH

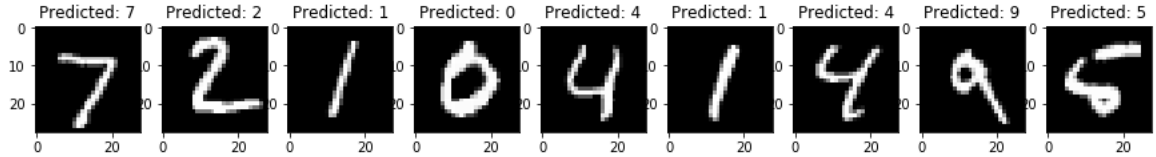

FIGURE 12. These are the loss, f1 score and accuracy score of the CNN after I applied all that I knew

3.5.2. **FMNIST**

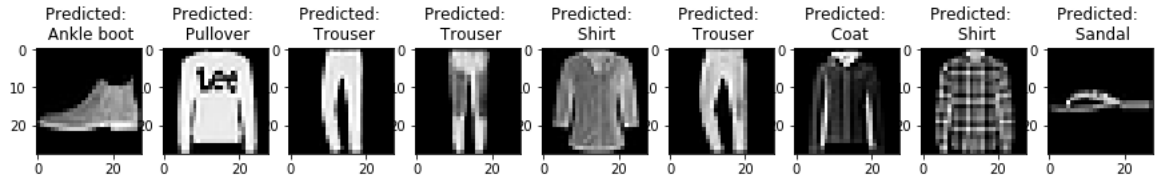

FIGURE 13. These are some of the correct classifications from the CNN.

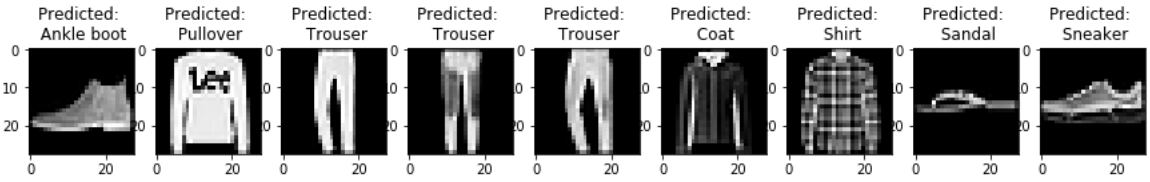

FIGURE 14. These are some of the correct classifications from the CNN-LSTM.

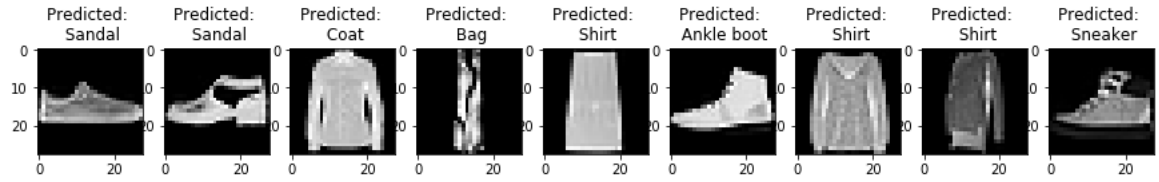

FIGURE 15. These are some of the incorrect classifications from the CNN.

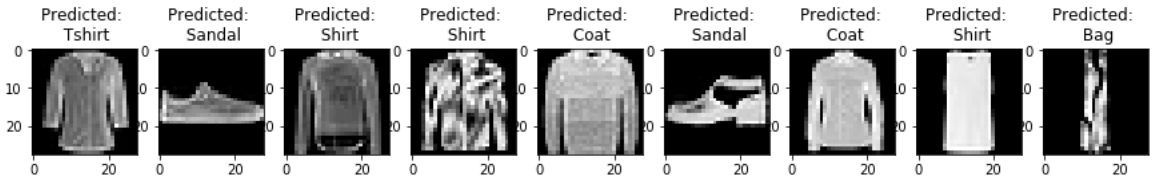

FIGURE 16. These images are some of the mislabeled images from the CNN-LSTM.

3.5.3. **MIML**

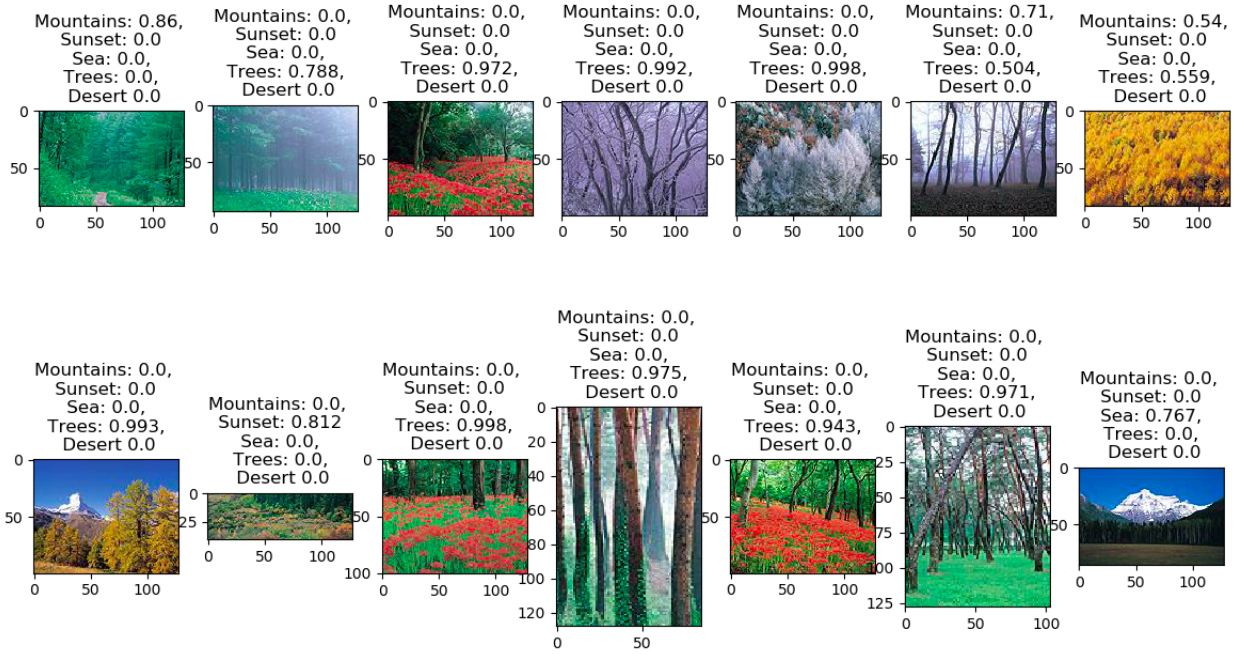

FIGURE 17. These are some of the classifications by the CNN

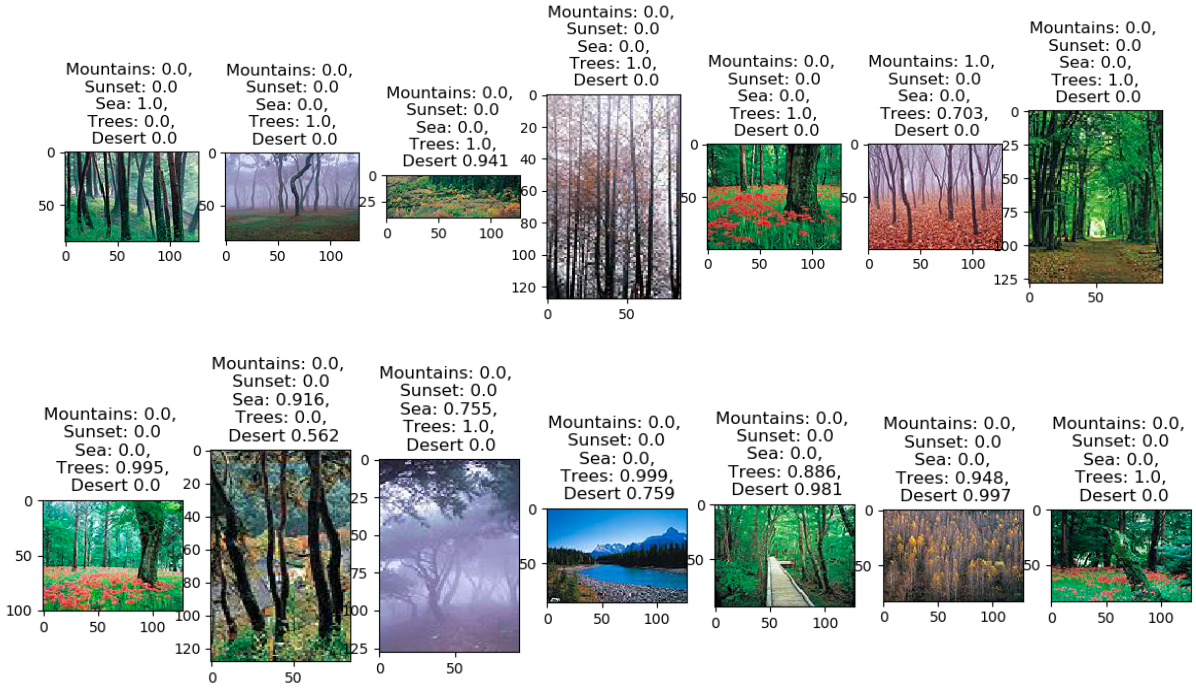

FIGURE 18. These are how the CNNLSTM saw some of the test images in the miml dataset.

3.5.4. **Human Protein Atlas**

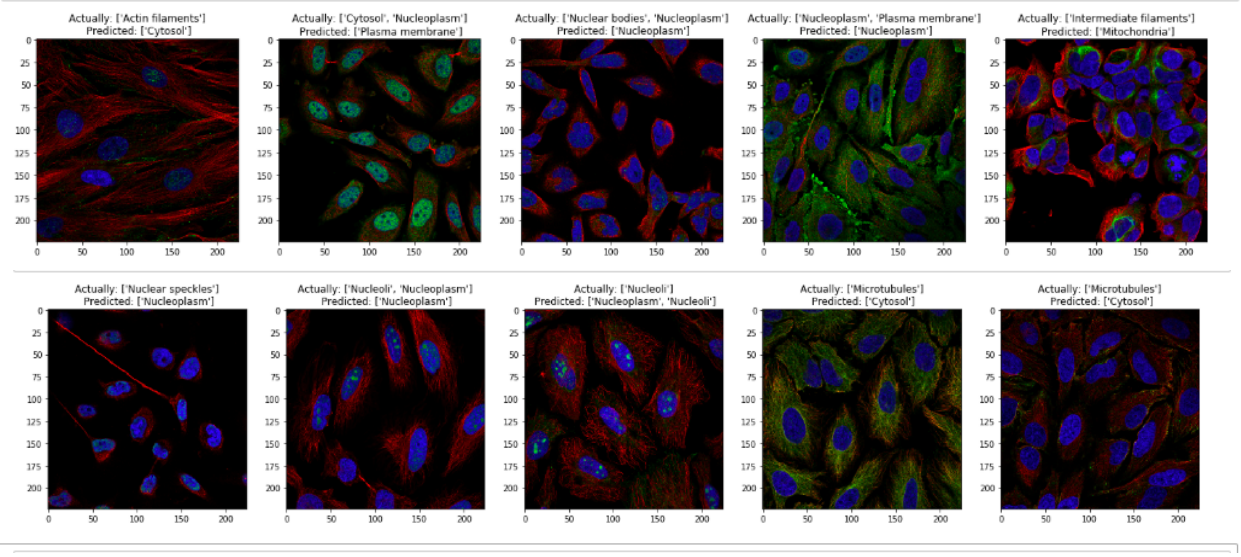

FIGURE 19. These are how the CNN saw some of the validation images in the Human Protein Atlas dataset.

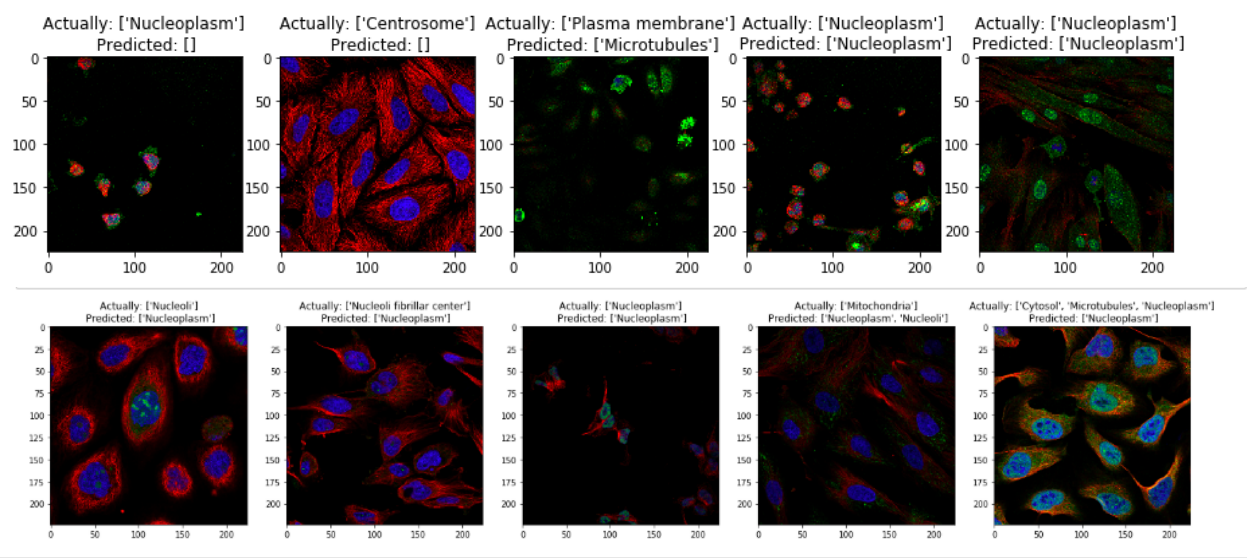

FIGURE 20. These are a sample of how the CNNLSTM saw the images of the validation set of the Human Protein Atlas dataset.

3.6. **Rationale for Evidence #3: Sample Classifications**

These images and their titles are meant to show how the CNN and the CNNLSTM did overall for each of the datasets that they were tested on. I included these classifications, as it is one of the best ways to see how each model did overall, and it allows you to analyze the models further. Throughout this process, it was a bit tricky to interpret what some of the calculated metrics and values meant. Sometimes I would get lost in these numbers and could not figure out why a model was acting a certain way. That is why, in order to truly analyze these models in a meaningful way, I would have to "see what they see". With this approach, I could see what labels each of the models struggled with and which of the labels each of the models was doing great with. With this new type of data, I could start to answer three of my fieldwork questions: *How does each model do on a multi-class dataset?*, *How does each model do on a multi-labeled dataset?*, and *How can these models be used in the real world?*.

For the FMNIST and MNIST datasets, these sample classifications allowed me to see what specific classes tripped up each of the models. For example, as Figure 1 shows, the CNN had

a hard time differentiating between the different versions of nines. This fact can be seen by the large amount of mislabeled nines in Figure 1. The CNNLSTM, however, had a hard time classifying images where the number was ambiguous or incomplete, rather than one class. This idea can be seen in the third, seventh and ninth images in Figure 2, where the 8 is not complete and the five looks like an S. With the FMNIST dataset, the CNN had a hard time differentiating between all of the different footwear in the dataset, while the CNNLSTM had a hard time differentiating between a shirt and a coat. These incorrect labelings show me that the CNNLSTM was better than the CNN for both multi-class datasets as it had fewer problems with fewer classes.

On the other hand, the sample classifications for the miml and proteins dataset allowed me to see how the CNN and the CNNLSTM saw each of the combinations of classes and whether or not it associated one class with another class. For example, Figure 9 shows that the CNN tended to associate large amounts of trees with mountains and would even classify some of the images as mountains rather than trees. Contrary to this, the CNNLSTM actually had an easier time finding trees, but could not really identify deserts well as it saw large amounts yellow in the image as a desert. For the proteins dataset, both networks could easily identify the nucleoplasms in the image, but the CNN was better at almost all of the other classes except for the Nucleoli.

3.7. **Evidence #4 Paper**

ANALYSIS

nature biotechnology

Possible Answer to Kaggle comp

Deep learning is combined with massive-scale citizen science to improve large-scale image classification

Devin P Sullivan[1,7], Casper F Winsnes[1,7], Lovisa Åkesson[1], Martin Hjelmare[1], Mikaela Wiking[1], Rutger Schutten[1], Linzi Campbell[2], Hjalti Leifsson[2], Scott Rhodes[2], Andie Nordgren[2], Kevin Smith[3], Bernard Revaz[4], Bergur Finnbogason[2], Attila Szantner[4] & Emma Lundberg[1,5,6]

Pattern recognition and classification of images are key challenges throughout the life sciences. We combined two approaches for large-scale classification of fluorescence microscopy images. First, using the publicly available data set from the Cell Atlas of the Human Protein Atlas (HPA), we integrated an image-classification task into a mainstream video game (EVE Online) as a mini-game, named Project Discovery. Participation by 322,006 gamers over 1 year provided nearly 33 million classifications of subcellular localization patterns, including patterns that were not previously annotated by the HPA. Second, we used deep learning to build an automated Localization Cellular Annotation Tool (Loc-CAT). This tool classifies proteins into 29 subcellular localization patterns and can deal efficiently with multi-localization proteins, performing robustly across different cell types. Combining the annotations of gamers and deep learning, we applied transfer learning to create a boosted learner that can characterize subcellular protein distribution with F1 score of 0.72. We found that engaging players of commercial computer games provided data that augmented deep learning and enabled scalable and readily improved image classification.

Analysis of large data sets is an increasingly important challenge[1]. Although machine learning, artificial intelligence and citizen science offer potential solutions to coping with this explosion of data[2–6], the large amounts of data that are generated as automated fluorescence microscopy systems become ever more widely used in quantitative biology create new challenges for automated image analysis.

The Human Protein Atlas (HPA) is an open-access database using antibody labeling and microscopy to systematically build an image-based map that details the spatial distribution of proteins in human cells and tissues (http://www.proteinatlas.org)[7]. Subcellular compartmentalization is fundamental to eukaryotic cells enabling multiple cellular processes to occur in parallel. The Cell Atlas in the HPA is building a proteome scale map of protein subcellular localization via hundreds of thousands of high-resolution confocal immunofluorescent images[8]. This map aids researchers in understanding protein function, interactions, cellular biology and, ultimately, disease. Given the magnitude of images continuously collected by the HPA Cell Atlas, a detailed analysis of the data requires a very large number of accurate image classifications.

Previous efforts to automate the classification of protein subcellular distribution from images have included methods such as k-NN classifiers, support vector machines, artificial neural networks and decision trees. Most studies have used hand-crafted feature sets[9–14], whereas others have used inference and multi-resolution techniques[15]. Recently, deep convolutional neural networks (CNNs) have been successful in classifying protein localization of single localizing proteins in budding yeast[16,17] and human cells[18]. These approaches, however, have focused on a limited number of single patterns (9–18 patterns), most often in a single cell type. This number of labels only provides a coarse description of biology as cellular architecture is more refined with specialized sub-organelle compartments and dynamic structures. However, the severe class imbalance introduced when considering rare cellular structures makes it harder to create a classifier that is capable of accurately predicting all localizations[19]. An even greater limitation to previous methods is that they only consider proteins in a single subcellular location, making them unsuitable for the ~50% of the human proteome that are multi-localizing[8]. Multi-localizing proteins are likely to be important for the inter-connectedness and adaptivity of cellular processes; thus, correctly localizing these proteins is key to our understanding of cell biology. Although methods of 'unmixing' a pair of known individual patterns have been put forward[20,21], to the best of our knowledge no global image-based subcellular protein classification method that handles multi-localizing proteins has been presented until now[22].

Crowd-sourced citizen science offers an alternative for large-scale image classification[6]. Projects such as FoldIt[23,24], Galaxy Zoo[25–27], EyeWire, EteRNA[28] and Quantum moves[29] represent implementations of citizen science in which large numbers of non-expert volunteers have contributed valuable scientific information. The major drawback of this approach is that implementing an engaging citizen

[1]Science for Life Laboratory, School of Engineering Sciences in Chemistry, Biotechnology and Health, KTH - Royal Institute of Technology, Stockholm, Sweden. [2]CCP hf, Reykjavik, Iceland. [3]Science for Life Laboratory, School of Computer Science and Communication, KTH - Royal Institute of Technology, Stockholm, Sweden. [4]MMOS Sàrl, Monthey, Switzerland. [5]Visiting appointment: Department of Genetics, Stanford University, Stanford, California, USA. [6]Visiting appointment: Chan Zuckerberg Biohub, San Francisco, San Francisco, California, USA. [7]These authors contributed equally to this work. Correspondence should be addressed to E.L. (emma.lundberg@scilifelab.se).

Received 24 December 2017; accepted 19 July 2018; published online 20 August 2018; doi:10.1038/nbt.4225

science project requires resources, knowledge and time that most laboratories lack. Furthermore, creating and maintaining an engaged user base is difficult in one-off citizen science projects. One method of dealing with this is paying for citizen science efforts, as in Amazon's mechanical turk (mturk)[30]; however, this method is prone to exploitation and low data quality[31].

Here we demonstrate two complementary and successful approaches for large-scale classification of protein localization patterns in microscopy images from the HPA Cell Atlas. The first utilizes the power of massive multiplayer online (MMO) games to create a new approach to citizen science and was a collaborative effort between the HPA, Massive Multiplayer Online Science (MMOS) and the video game developer CCP Games. This partnership substantially reduced the effort to the lab by allowing CCP Games to develop the interface and MMOS to handle data management and serving. The result was the scientific project of image classification seamlessly integrated into the EVE Online universe, an MMO science fiction game with ~500,000 active players each month. The resulting mini-game, Project Discovery (PD), was successful in terms of participation, player retention, number of images classified and accuracy. In the second approach, we present Loc-CAT, a model for automated image classification of subcellular protein distribution patterns using deep neural networks (DNNs). To the best of our knowledge, this method represents the first tool for classifying protein distribution in human cells in microscope images capable of predicting robustly across cell types for proteins with an unknown number of locations. Furthermore, we compared the performance of the respective approaches and found that the gamer output could be used to improve deep learning models. Altogether, both approaches provide a refinement of the biological details in the HPA Cell Atlas. We believe that integration of scientific tasks into established computer games can be a valuable approach in the future with the power of rapidly leveraging the output of large-scale science efforts.

RESULTS

Subcellular distribution of proteins in microscopy images

Each sample in the HPA Cell Atlas consists of human cells that are immunofluorescently labeled for one protein of interest and three reference markers: DAPI for the nucleus and antibody-based labeling of microtubules and the endoplasmic reticulum. High-resolution images were acquired using confocal microscopy (**Fig. 1a**). The resulting images were annotated to determine the localization(s) of the protein of interest with the help of the three cellular reference markers. This study was performed using the Cell Atlas of the Human Protein Atlas version 14.0 (HPA Cell Atlas v14; **Supplementary Data Set 1**) in which protein distributions were classified into 20 organelles and subcellular structures (**Fig. 1b**). To refine the biological details of the HPA Cell Atlas, players in PD were asked to re-classify these images and classify the protein distribution into an additional ten patterns (**Fig. 1c**,**d**), for a total of 29 patterns in 17 human cell lines (**Supplementary Table 1**).

Some protein localization patterns, such as the centrosome, were small and easily overlooked, whereas others, such as the cytokinetic bridge, occurred in only a small fraction of cells. Adding to the complexity, some compartments, such as actin filaments, the Golgi apparatus and mitochondria, displayed highly heterogeneous morphologies across cell lines, making them more difficult to recognize (**Fig. 1e**). Class frequency (that is, protein localization) varied widely in human cells, from 0.016–24.3% in the HPA Cell Atlas v14. Furthermore, ~50% of proteins were localized to multiple cellular compartments[8] (**Fig. 1f**). Cell-to-cell variability could create further confusion (**Fig. 1g**). Together, these findings demonstrate that location classification is hardly a trivial task.

Image classification by citizen science task in EVE Online

In PD, players in EVE Online performed the aforementioned protein image classification. This project represents the first time a scientific task has been directly and seamlessly integrated into a mainstream video game narrative (**Fig. 2a**). The resulting mini-game was accessible from anywhere and at any time in the virtual universe of EVE Online. Participants were trained using a small set of preselected images gradually increasing in difficulty. Classification options were initially restricted to ease players into the complexity of the task. Participants in PD were motivated with leveled badges and in-game currency with which they could purchase exclusive items. This approach was able to easily gather and maintain participants, something other citizen science projects have struggled with, as measured by 'project appeal'[32] (**Fig. 2b** and Online Methods). This participation drive can particularly be seen when EVE became free to play, causing an increase in PD participation (day 250; **Fig. 2c**).

Participation peaked on day 3, with 5,507 players contributing 292,374 classifications (**Fig. 2c**). In total, 322,006 players of EVE Online played PD and contributed ~33 million image classifications. Of these, 59,901 players passed the training and tutorial phases and had above threshold performance, leading to 23.7 million high-quality image classifications. From this set of 59,901 players, on average 6,846 unique players contributed each month with a 30-d monthly retention rate of 32% (68% churn), and a rolling retention of 53% (47% churn) over the first 6 months, which was very good compared with other in-game features over the same period and vastly improves on previous citizen science efforts (**Supplementary Fig. 1**)[33].

Measuring player performance

To assess data quality, we used the F1 score, a measure of accuracy suitable for multi-label data, with the HPA Cell Atlas v14 image labels as ground truth. Initially, players received an additional reward for agreeing with the eventual community consensus. This reward was quickly exploited by gamers converging to a single annotation (cytoplasm, day 0–20; **Fig. 2d**) and was therefore removed. This resulted in a rapid improvement of accuracy (**Fig. 2d**). On the basis of player feedback, we created and implemented a larger set of more difficult control images including multi-localizing proteins and image artifacts. This led to a significant increase in data quality (day 50, $P < 4 \times 10^{-70}$, day 0–50 versus 50+, two-tailed t test; **Fig. 2d**).

To guard against erroneous annotations, we required a minimum of 12 votes per image before evaluating each task for a consensus using a hypergeometric test. Consensus was considered to be reached only if the number of votes for at least one class was significantly greater than would be expected at random ($P < 0.01$) and no other classes were near the decision threshold ($P < 0.1$). If consensus was not reached, the task was kept open and more votes were acquired. On average, each task required 15 player votes (median = 13) to reach a consensus. Given the speed of players, the data set was annotated six times, resulting in a median of 78 annotations per image. This statistical approach, together with the high number of annotations per image, allowed us to tolerate annotations from players performing worse than naively guessing the single most common class, accounting for ~10% of the annotations (**Fig. 3a**). The overall F1 score was 0.55 with a mean per-class F1 score of 0.50. In general, gamers performed better for common categories, presumably because they are more accustomed to these. Microtubules were a notable exception, as the gamers had a reference channel, allowing them to easily recognize this pattern (F1 = 0.78).

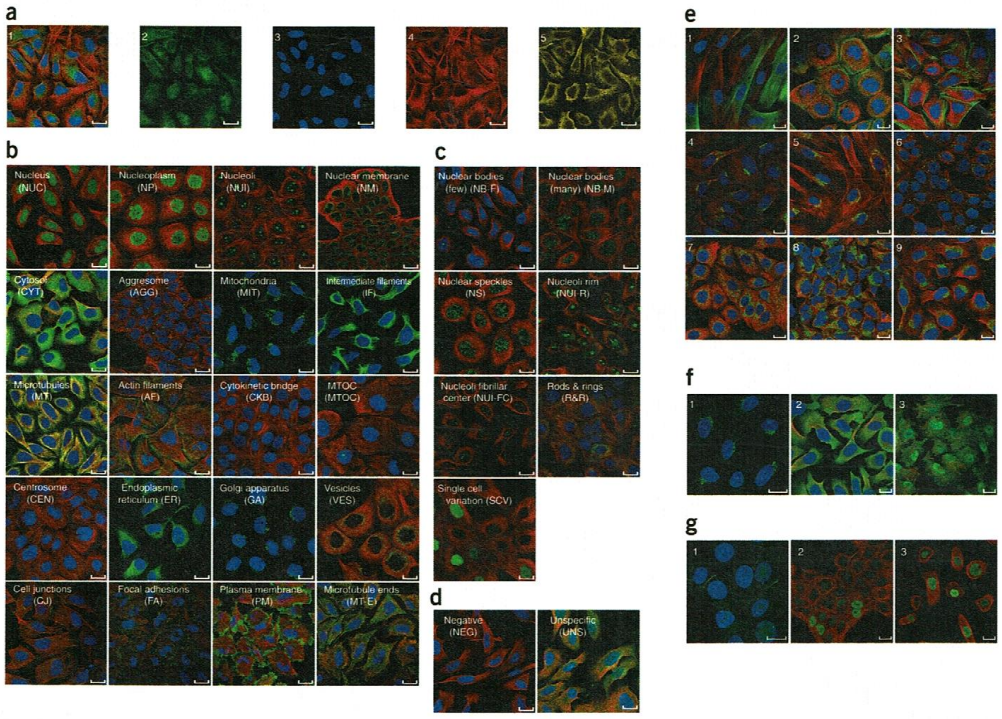

Figure 1 Illustrative data from the HPA Cell Atlas. The HPA Cell Atlas contains four-channel confocal images for the majority of all human proteins. Scale bars represent 20 µm (inner length). The experimental procedure was described previously[8]. (**a**) Example composite image (1) consisting of false-colored channels representing the protein of interest (green, 2), with DAPI labeling the nucleus (blue, 3), an antibody labeling the microtubules (red, 4). Each assay also contains an antibody labeling the endoplasmic reticulum (yellow, 5). (**b**) The images in HPA Cell Atlas v14 were classified into 20 organelle patterns of major organelles. (**c**) In PD, players classified seven additional patterns. (**d**) Two additional patterns were classified to filter negative and unspecific experiments. (**e**) Organelles displayed morphological differences across cell lines such as actin filaments (1–3), Golgi (4–6) and mitochondria (7–9). (**f**) Multi-localizing proteins offer another challenge for accurate pattern classification. Often patterns were hard to distinguish when occurring together, such as Golgi and vesicles (1), or cytoplasm and plasma membrane (2). Some localizations were not visible or easily distinguishable in every field of view, particularly when they occurred in variable focal planes. For example, although focal adhesions, nucleoplasm, and cytosol were in focus, a Golgi apparatus localization in another focus plane could not be seen (3). (**g**) Cell-to-cell variations can be challenging and viewing images with various channel combinations can aid annotations of patterns such as cell junctions (1) or nucleoplasm (2 and 3).

PD ace can't move up v/o change

An alternative would be expectation maximization for jointly estimating player bias and protein localization[34]. This approach was not feasible during gameplay given the computation time, and did not perform as well as the aforementioned consensus approach in *post hoc* analysis (**Supplementary Table 1**). As a result of the large number of annotations per image, it is also unlikely that additional annotations would improve the accuracy of PD without a shift in player behavior (retraining) or task (sub-set annotation)[35]. This is supported by the lack of correlation between the number of images analyzed, time-on-task per player and performance (**Supplementary Fig. 2**).

The frequency at which players co-annotated classes was compared with the co-annotation frequencies in the HPA Cell Atlas v14 using independent Bonferroni corrected binomial tests to estimate multi-label confusion for each class (**Fig. 3b** and **Supplementary Fig. 3a**). Patterns of structurally similar organelles appeared to be frequently confused, such as centrosomes and microtubule organizing centers (**Fig. 3c**). Although much rarer, confusion across the nuclear and cytoplasmic spaces could also be observed, for example, when gamers annotated vesicles instead of nuclear bodies, both of which are dot-like structures.

Players were also able to report unusual findings in images. A review of all images with more than 20 such reports mainly revealed rare cellular morphologies such as blebs and membrane protrusions, or staining artifacts. However, this demonstrates that the players are capable of finding patterns that deviate from the common patterns, and identified several interesting and previously unannotated patterns such as vesicle fronts and condensed chromosomes.

Adjusting for PD player bias leads to improved data quality

Bias messed up Data

Of the 29 patterns classified in PD, 20 were previously annotated in the HPA Cell Atlas v14 and 3 additional patterns were annotated

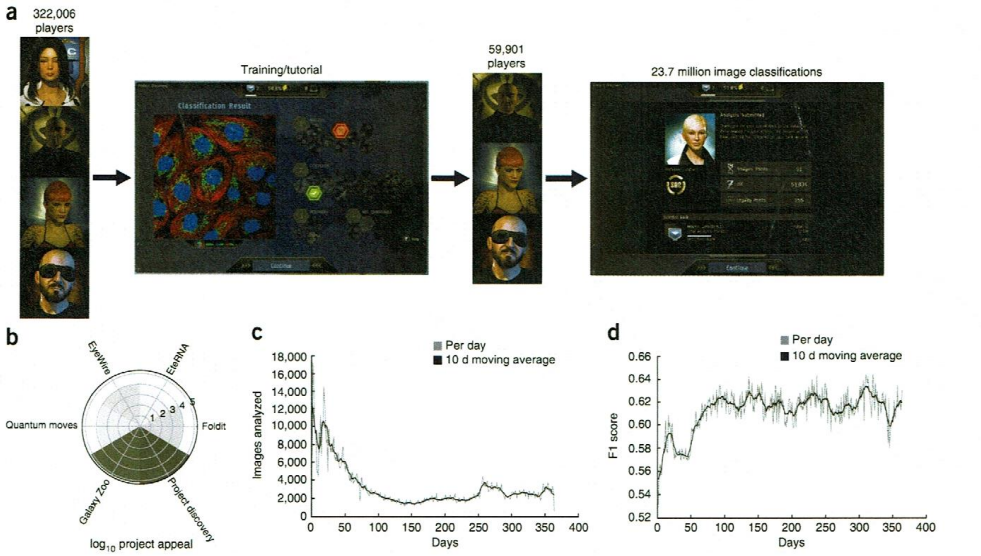

Figure 2 PD workflow. (**a**) Nearly 350,000 players in EVE Online played PD. Before contributing classifications, players were trained via an interactive tutorial with an increasing number of classes available and increasingly difficult samples. Once they passed the training phase, close to 60,000 players went on to contribute over 23.7 million image classifications. (**b**) The $\log_{10}$ project appeal, defined as (number of volunteers)/(project time^2), demonstrates the way that PD participation dwarfed previous citizen science efforts (gray indicates project). (**c**) Participation peaked on day 3, with over 17,000 images analyzed (minimum 12 players per image); however, throughput decreased over the first 150 d of the project (per day, gray; 10-d moving average, black). Participation substantially increased around day 250 of the project, when EVE Online went free to play, demonstrating that in-game mechanics can be directly used to influence participation. (**d**) Data quality also took time to stabilize, and removing rewards for community consensus agreement, together with new control samples integrated around day 50, significantly improved accuracy ($P < 4 \times 10^{-70}$, day 0–50 versus 50+, two-tailed t test).

internally while PD was active (nucleoli fibrillar center, nuclear speckles and nuclear bodies). This allowed us to examine the annotation trends of the players. After initial assessment, it was clear that some classes such as cytoplasm, nucleus and vesicles were over-annotated (**Fig. 4a**). To correct for this bias, we used the class distribution in the reference HPA Cell Atlas v14 data set to create per-class cutoffs

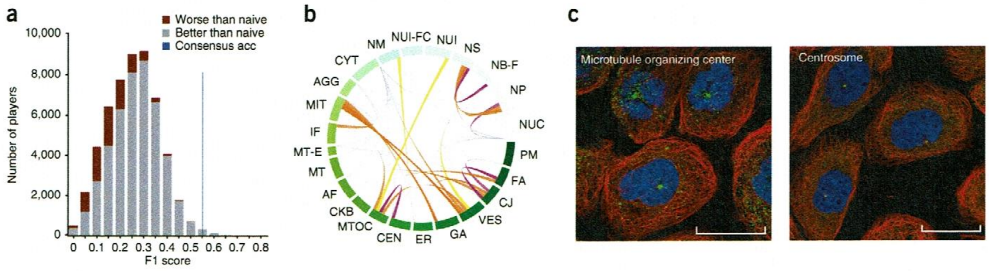

Figure 3 PD performance and confusion. (**a**) F1 score distribution of players with a minimum of ten analyzed images ($n = 59{,}901$). Despite nearly 10% of players performing worse than naively guessing the most common class (nucleoplasm, maroon bar sections), the consensus accuracy (blue line, 65,596 hypergeometric test consensuses, $P < 0.01$, $n \geq 12$ per test) remained markedly higher than the player average. (**b**) Circular plot showing player over-represented co-annotations with solution classes from the HPA Cell Atlas v14 ($P < 10^{-3}$, Bonferroni corrected one-tailed Binomial test, per-class sample sizes are found in **Supplementary Fig. 3**) serving as a proxy for multi-label confusion. Bars for each class are scaled to $\log_2$ class size and color (green) is used for visual clarity. Classes containing more than five over-represented co-annotations are considered 'uniformly over-annotated' and are shown in gray (NUC, CYT, MT-E and AGG). Colors represent the distance between classes (nodes) in the hierarchical structure, with purple, salmon and yellow representing $d = 2$, 4 or 6, respectively. Over-annotations were directional, with the thick end of the ribbon indicating which class was over-annotated by players (confused with) relative to the HPA Cell Atlas v14. Ribbons with two thick ends indicate a bi-directional over-representation of the co-annotation. (**c**) Centrosome ($n = 1{,}498$ images) and MTOC ($n = 424$ images) is an example of a bi-directional over-representation co-annotation. Scale bars represent 20 µm (inner length).

Per class increased by .3 overall moved by 0.13

(**Supplementary Table 2**). This correction is not possible for data where the approximate proportions of classes are unknown. This approach led to a large improvement in per-class F1 score for over-annotated classes such as cytosol and nucleus, resulting in an average per-class F1 score of 0.53 (**Fig. 4b**) and an overall mean F1-score of 0.68. This includes novel classes for which we chose the most permissive cutoff to maximize discovery (recall).

Refined classifications of images in the Cell Atlas

A major contribution of the participants in PD was to refine the classifications in the Cell Atlas. Although work is still underway to incorporate all this data, version 18 of the HPA Cell Atlas includes five new categories annotated by the gamers, in total refining localization information for 2,902 proteins. Gamer annotations of classifications such as 'nucleoli fibrillar center' or 'nucleoli rim' were nearly entirely contained in their previously annotated parent class 'nucleoli' (99%), indicating that many of these annotations are indeed refining annotations in v14 (**Supplementary Fig. 3b**). Cytoophidium, or Rods and Rings (R&R), are an excellent example of a fairly uncharacterized transient cellular structure, with only three previously known protein members[36,37]. In addition to the known component protein encoded by IMPDH1, the gamers identified ten additional R&R proteins that were confirmed by colocalization analysis to localize to R&R after induction with ribavirin (**Supplementary Table 3**), such as UPF0769 protein C21orf59 (**Fig. 4c**). By expanding the set of known R&R proteins, players in PD have shed new light on this structure that may help in understanding its biological function.

PD refined labels

Image classification with Loc-CAT using deep learning

Another approach for classification of image patterns is machine learning. Toward this end, we used a deep neural network to create Loc-CAT. Inputs for this network were previously optimized subcellular localization features (SLFs; **Supplementary Data Set 2**) calculated on segmented single cells[9,20]. As with PD, the ground truth was the labels for images in the HPA Cell Atlas v14. Predictions by Loc-CAT were made per-cell. The mean per-cell predictions were then used to create a per-image classifier on which class-specific decision boundaries were adjusted in a parameter tuning step (**Supplementary Table 4**). Along these lines, a major challenge with this approach is recognizing patterns that may occur in only a few cells in the image and discerning them from a false-positive prediction, such as the cytokinetic bridge, aggresome, centrosome and microtubule organizing center (**Fig. 5a**). Another challenge when classifying segmented cells is to recognize patterns at the cell periphery, such as plasma membrane, focal adhesions and cell junctions. Representative samples for classes with poor performance demonstrate that Loc-CAT struggles to recognize cell-to-cell variable patterns and patterns at the cell periphery (**Fig. 5b**).

Feature vector → Prediction

my Prob? ←

Ignores sides

Most previous methods[12–14,17,38] have controlled for biological variance by restricting classification to a single cell line. To test the robustness of Loc-CAT, we trained models on all 17 of the cell lines present in the HPA Cell Atlas v14 individually and applied each model to each cell line in turn (**Fig. 5c**). On the basis of this comparison, we can conclude that the performance of Loc-CAT was significantly higher when training on cell lines with more data ($P < 10^{-10}$, two tailed t test). The generalized model trained on all of the cell lines was capable of predicting subcellular localization across variations in morphology with high accuracy and performed best on nearly all of the cell lines tested.

Previous methods have also been limited to single-label predictions. To test the performance of Loc-CAT in classifying single labels, multiple labels and mixed labels (data set containing both single and multi-label images), we trained models on these groups separately and applied them to each group in turn (**Fig. 5d**). Loc-CAT significantly improved localization accuracy when predicting on multi-label or mixed single and multi-label data relative to a comparable single-label-based approach ($P < 10^{-4}$, two-tailed students t test), indicating that this method is more generally applicable for images where the number of localizations is not known a priori.

→ GAB

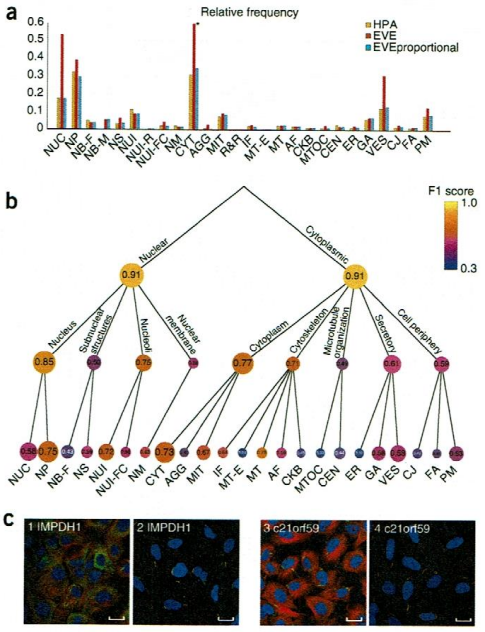

Figure 4 Correcting player bias. To improve data quality, the proportions of each class in the HPA Cell Atlas v14 were used to tune the significance levels for each class. (**a**) Bar plot showing relative proportion of data with an annotation in each class before (red) and after (blue) tuning as compared with the HPA Cell Atlas v14 (yellow). The bar for cytosol extends off the graph to a value of 0.73 before tuning (red bars). (**b**) Evaluation of classes in a tree-based hierarchy allowed evaluation of confusion between functionally and spatially similar patterns (ball size = $\log_{10}$ class size). As the depth in the tree decreased, cellular compartments were merged into meta-compartments. Votes from leaves were pooled before calculating consensus for each meta-compartment, showing player performance at different granularities. Vote pooling results in a stricter cutoff for each meta-class and, for this reason, nuclear membrane accuracy dropped in the second layer of the tree despite there being no branch. (**c**) Participants in PD shed new light on the little-known R&R structure, including correct identification of the IMPDH1 encoded protein known to localize to R&R and the novel discovery of the localization to R&R of the UPF0769 protein C21orf59. For each protein (green), localizations were verified by an independent colocalization study with a marker for R&R (red, 2, 4) after induction of R&R formation with ribavirin. Colocalization assays (2, 4) are shown beside the standard HPA Cell Atlas image (1, 3). High colocalization (yellow) confirmed that these proteins (green) were localized R&Rs (red). These experiments were performed in triplicate.

Evaluation of Loc-CAT and citizen science performance

Despite the high performance of Loc-CAT, players in PD (average per-class F1 = 0.53) outperformed Loc-CAT (average per-class

PD: better @ rarer classes Loc-CAT = great @ big trends

ANALYSIS

F1 = 0.47), particularly in many of the less common classes, for example, microtubule ends, which has only 32 images. Loc-CAT outperformed PD in most other classes, particularly on classes with large amounts of training data and endoplasmic reticulum (ER) where Loc-CAT has access to an additional reference channel players in PD did not (**Fig. 5a**). This makes the two methods closer in performance when comparing

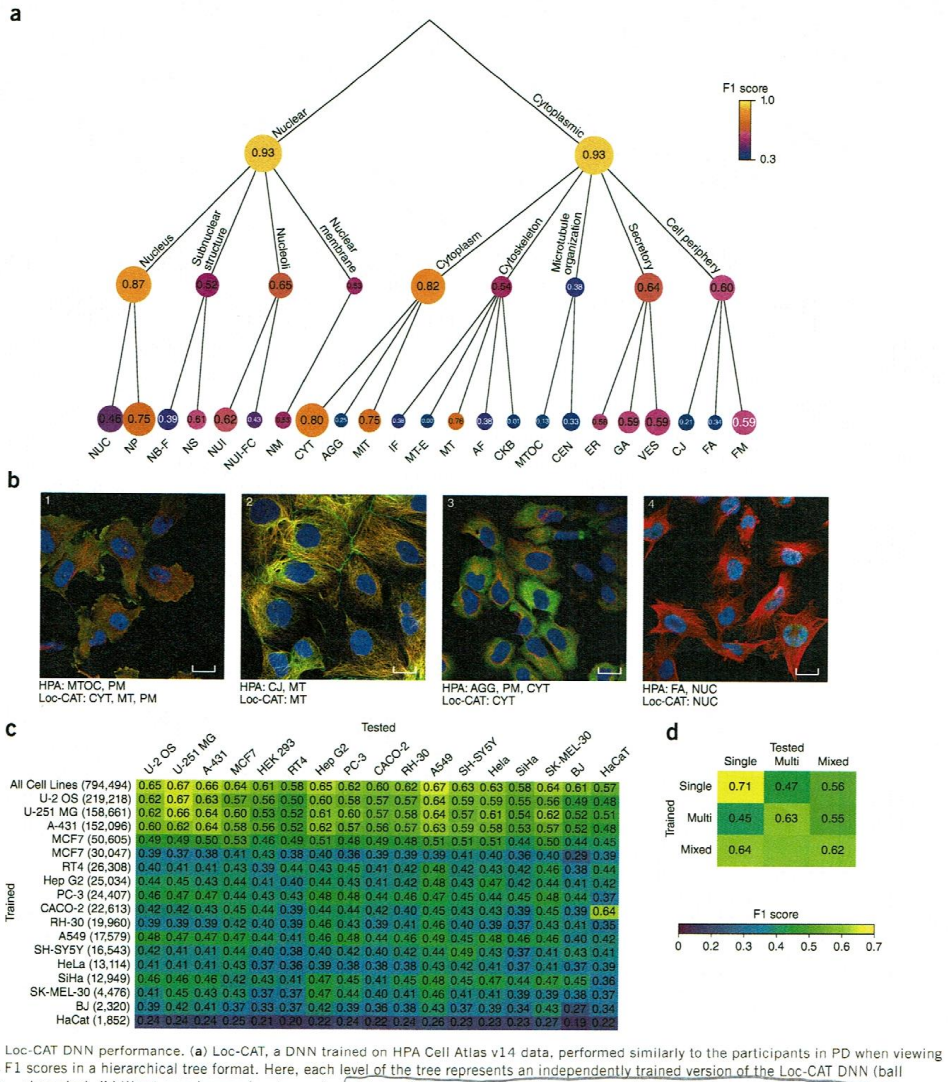

Figure 5 Loc-CAT DNN performance. (**a**) Loc-CAT, a DNN trained on HPA Cell Atlas v14 data, performed similarly to the participants in PD when viewing per-class F1 scores in a hierarchical tree format. Here, each level of the tree represents an independently trained version of the Loc-CAT DNN (ball size = $\log_{10}$ class size). (**b**) Worst-case images for classes in which Loc-CAT performed poorly are shown with the HPA annotation and Loc-CAT DNN annotation (images selected from false-negative rank list, lowest confidence in class of interest). Scale bars represent 20 μm (inner length). (**c**) Cross cell-line performance (average F1, fivefold cross validation) of Loc-CAT for the 17 cell types present in the HPA Cell Atlas v14. Number of cells used in training are shown in parentheses for each data set. (**d**) Loc-CAT networks trained (*y* axis) versus tested (*x* axis) on single-label (n = 55,083 cells, 5,234 images), multi-label (n = 58,133 cells, 5,560 images) and a mixture of single and multi-label data (n = 57,320 cells, 5,416 images, 51% multi-label) using U-2 OS cells from the HPA Cell Atlas v14 demonstrate that a model trained on a mixture of single and multi-localizing proteins generalizes best to novel data in which the number of locations a protein is present in are not known a priori.

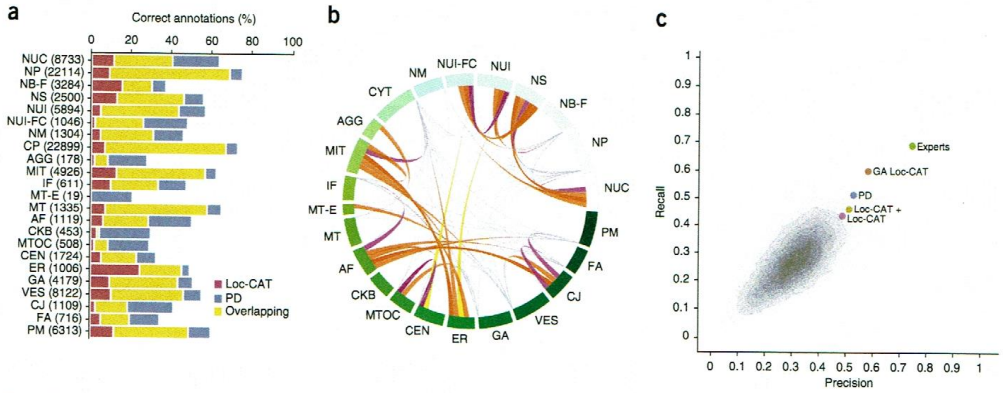

Figure 6 Transfer learning boosts Loc-CAT DNN performance. True positive percentages (recall) for each class identified by Loc-CAT (pink), PD (purple) and both (overlapping, yellow). (**b**) Loc-CAT over-represented co-annotations with solution classes from the HPA Cell Atlas v14 ($P < 10^{-3}$, Bonferroni corrected one-tailed Binomial test, per-class sample sizes are found in **Supplementary Fig. 5**) serves as a proxy for multi-label confusion. Bars for each class are scaled to $\log_2$ class size and color (green) is used for visual clarity. Classes containing more than five over-represented co-annotations were considered to be 'uniformly over-annotated' and are shown in gray. Given that over-annotations are directional, tapering was used to indicate directional confusion. Colors represent the distance between classes (nodes) in the hierarchical structure with purple, salmon and yellow representing $d = 2$, 4 or 6 respectively. Over-annotations were directional, with the thick end of the ribbon indicating which class was over-annotated by Loc-CAT relative to the HPA Cell Atlas v14. Two thick ends indicate a bi-directional over-representation. (**c**) Average per-class precision versus recall plot on HPA Cell Atlas v14 (all cell lines). PD bias corrected consensus (purple) compared with players in PD (gray points, contours). Loc-CAT (pink) performance was dragged down by low-frequency classes; however, transfer learning using both PD player input and computational features (GA Loc-CAT, red) outperformed both Loc-CAT and the PD results. Generation of 'pseudo-gamer' data for use in the transfer learner when player data was not available improved Loc-CAT (Loc-CAT+, gold); however, experts in the HPA Cell Atlas (experts, green) still vastly outperformed all of the other methods.

overall F1 score (Loc-CAT = 0.65, PD = 0.68). PD continued to outperform Loc-CAT when examining the middle layer of resolution in the organelle hierarchy (**Figs. 4b** and **5a**). Notably, gamers appeared to be more accurate at identifying nucleoli-related patterns and continued to outperform Loc-CAT in the cytoskeleton and microtubule organization meta-classes.

Loc-CAT and PD were also evaluated relative to previous methods for classification of localization patterns in images (**Supplementary Table 1**). A direct comparison was made by testing the proposed methods on the lower-complexity single-label data set used in other studies. Despite being trained on over twice as many classes, Loc-CAT was able to predict protein localization in these images with nearly equivalent per-class precision and recall as previous methods trained on this data set. PD was substantially better than all of the methods in per-class recall and overall precision, but struggled with some classes, lowering the per-class precision. In addition, a convolutional architecture based on SimpleNet was introduced to Loc-CAT instead of using traditional image features[39]. Although other convolutional architectures may perform well, this approach did not outperform the SLFs used in this work (**Supplementary Table 1**).

Gamer augmented transfer learning improves Loc-CAT accuracy

Although the overall accuracies of PD and Loc-CAT are relatively similar (**Figs. 4b** and **5a**), per-class true-positive overlap revealed that correctly annotated images varied widely (**Fig. 6a**). This suggests that labels generated in PD represent a substantial amount of per-image information in addition to the five novel classes. To leverage this information, we applied a transfer-learning approach in which we fed gamer annotations as a set of additional input features to Loc-CAT, resulting in increased performance (GA Loc-CAT; **Fig. 6c**). Because we will not have gamer input for all future tasks, we extended this approach by training a shallow 'pseudo-gamer' network (**Supplementary Fig. 4**). The resulting pseudo-gamer predictions were then fed into the Loc-CAT DNN as additional input features. This combined network, henceforth referred to as Loc-CAT+ (**Fig. 6c** and **Supplementary Fig. 4**), displays many of the same overrepresented co-annotations (**Fig. 6b** and **Supplementary Fig. 5**) as players in PD (**Fig. 3c**). Notably, however, overrepresented co-annotations between major compartments (**Figs. 3c** and **6b**) differed between the two approaches. For example, Loc-CAT+ annotates endoplasmic reticulum together with nucleoli more frequently than expected, a behavior that was not seen by the gamers. Nevertheless, the Loc-CAT+ model allowed us to incorporate some of the insights of the gamers, improving the performance of Loc-CAT by raising the average per-class F1 score from 0.44 to 0.47. However, experts in the HPA Cell Atlas (**Fig. 6c**) still outperformed all of the methods in a randomized blind annotation test (per-class F1: 0.71, overall F1: 0.76; **Supplementary Data Set 3**), suggesting that there is room for further improvement in computational image classification.

DISCUSSION

This work presents two complementary approaches to high-throughput classification of subcellular localizations in fluorescent microscope images from the HPA Cell Atlas. Multi-localizing proteins, large class imbalance, cell line variations and rare patterns that may not be present in all of the cells in an image make annotation of this dataset challenging.

The first approach uses the power of MMO games through the PD mini-game in EVE Online to perform large-scale image classification. This is the first implementation of a scientific task into an existing

mainstream video game. This approach reduces development costs to labs for citizen science and demonstrates that players in MMO games can produce high-quality data despite potentially being motivated by alternative in-game dynamics or fun, rather than connection to a cause. An equivalent annotation using mechanical turk and a reward of $0.01–0.05 per task would result in costs of $0.33–1.65 million to obtain an equal number of annotations in addition to requiring the same effort in preparation, data management and analysis. This approach also solves issues surrounding the creation and maintenance of a user base in citizen science, as in-game rewards can be used to drive participation.

Training of players proved to be important for obtaining good results. The initial training images were too simple relative to the general population, and player performance improved significantly when more challenging training images were introduced ($P < 4 \times 10^{-70}$, day 0–50 versus 50+, two-tailed t test). Vote aggregation and statistics allowed us to tolerate noise in player annotations, and basic knowledge of the background distribution of classes allowed us to mitigate the effects of player bias. In future efforts, simplifying the task (for example, binary classification for the presence of a single class) may improve accuracy in a cost-effective manner, as throughput is not a large concern for this gamification paradigm. Through PD, players assisted in the refinement of annotations for thousands of samples, including several members of the largely uncharacterized R&R structure.

Participation in PD on behalf of the gaming company (CCP games) is voluntary based on their desire to promote scientific research and foster good will in their player base. This approach was highly rewarding and is promising for other massive analysis problems, with a major caveat being that the data set needs to be large, as the players were very fast. In addition to providing-high throughput image analysis, scientific outreach was a huge benefit of this method, reaching a broad community that is not necessarily invested in science. Future projects can further benefit from the development of the PD citizen science platform, even across disciplines, as exemplified by the recently launched Project Discovery Exoplanets in EVE Online.

Although PD represents one of the most successful citizen science efforts to date, it relies on continuous manual efforts of many participants administered by a third party and is therefore not sustainable for long-term generalized future use, as the gaming company may decide to take down the game. For this purpose, our DNN-based approach, Loc-CAT+, provides a promising method for annotating protein localizations in future work as it is fully automated. Loc-CAT+ represents major improvements on previous efforts, as a result of its ability to accurately classify a large number of patterns and mixtures thereof, as well as generalize across cell types with different morphologies. Thanks to this generalizability and ability to classify multi-localizing proteins, it is, to the best of our knowledge, the first automated image-based protein localization method capable of accurately classifying images where no information about the protein is known a priori. Furthermore, by augmenting the quantitative image features used in Loc-CAT with PD gamer annotations we improved Loc-CAT to nearly human performance. One major challenge in machine learning remains the recognition of rare and novel classes in which there is little or no training data. In our study, humans still clearly outperformed the algorithmic approaches. The refinement of the annotations in the HPA Cell Atlas made through PD and Loc-CAT, with the novel classification of seven additional subcellular localizations, present an exciting new resource for understanding cell biology. Although preliminary tests of convolutional neural networks in this work did not improve results over the quantitative image features used, different model architectures and hyperparameters may provide the improvements needed to reach expert performance.

To summarize, we demonstrated two alternative approaches for large-scale classification of protein distribution patterns in microscopy images. Furthermore, we showed how gamers and DNNs excel at different types of classifications and that gamer output can be used to augment and improve deep learning models. Finally, we speculate that the integration of scientific tasks into established computer games will be a commonly used approach in the future to harness the brain processing power of humans and that intricate designs of citizen science games feeding directly into machine learning models through techniques such as reinforcement learning have the power of rapidly leveraging the output of large-scale science efforts.

METHODS

Methods, including statements of data availability and any associated accession codes and references, are available in the online version of the paper.

Note: Any Supplementary Information and Source Data files are available in the online version of the paper.

ACKNOWLEDGMENTS

We acknowledge the staff of the Human Protein Atlas program for valuable contributions. We acknowledge the EVE Development team, the University of Reykjavik and the University of Iceland for assistance with the game implementation. We acknowledge MMOS Sarl for serving images and managing response collection and CCP hf and MMOS Sarl for financially supporting the image storage and serving throughout Project Discovery. Funding to E.L. was provided by the Knut and Alice Wallenberg Foundation. Finally we acknowledge all EVE Online players that participated in Project Discovery!

AUTHOR CONTRIBUTIONS

A.S., B.R., B.F., A.N. and E.L. conceived the study. M.H., A.S., B.F., E.L., D.P.S. and C.F.W. developed the methodology for the study. A.S. and B.R. developed the citizen science engine. L.C., H.L., S.R. and B.F. developed the game narrative and implementation. Project Discovery was played by thousands of players of EVE Online. D.P.S., L.Å., M.W., R.S. and E.L. provided game support. C.F.W., K.S. and D.P.S. developed the machine learning. D.P.S., C.F.W. and E.L. carried out data analysis and investigation. D.P.S., C.F.W. and E.L. wrote the manuscript. D.P.S. and C.F.W. created the figures. E.L. supervised and administered the project and acquired funding.

COMPETING INTEREST

A.S. and B.R. are founders of MMOS Sarl.

1. Bouwer, J. *et al.* Petabyte data management and automated data workflow in neuroscience: delivering data from the instruments to the researcher's fingertips. *Microsc. Microanal.* **17**, 276–277 (2011).
2. Ferrucci, D. *et al.* Building Watson: an overview of the DeepQA project. *AI Magazine* **31**, 59–79 (2010).
3. Larrañaga, P. *et al.* Machine learning in bioinformatics. *Brief. Bioinform.* **7**, 86–112 (2006).
4. Silver, D. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
5. Litjens, G. *et al.* A survey on deep learning in medical image analysis. *Med. Image Anal.* **42**, 60–88 (2017).
6. Cohn, J.P. Citizen science: can volunteers do real research? *Bioscience* **58**, 192–197 (2008).
7. Uhlen, M. *et al.* Towards a knowledge-based Human Protein Atlas. *Nat. Biotechnol.* **28**, 1248–1250 (2010).
8. Thul, P.J. *et al.* A subcellular map of the human proteome. *Science* **356**, eaai3321 (2017).
9. Boland, M.V. & Murphy, R.F. A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of HeLa cells. *Bioinformatics* **17**, 1213–1223 (2001).
10. Huang, K. & Murphy, R.F. Boosting accuracy of automated classification of fluorescence microscope images for location proteomics. *BMC Bioinformatics* **5**, 78 (2004).
11. Newberg, J.Y. *et al.* Automated analysis of Human Protein Atlas immunofluorescence images. *Proc. IEEE Int. Symp. Biomed. Imaging* **5193229**, 1023–1026 (2009).
12. Li, J., Newberg, J.Y., Uhlén, M., Lundberg, E. & Murphy, R.F. Automated analysis and reannotation of subcellular locations in confocal images from the Human Protein Atlas. *PLoS One* **7**, e50514 (2012).

13. Li, J., Xiong, L., Schneider, J. & Murphy, R.F. Protein subcellular location pattern classification in cellular images using latent discriminative models. *Bioinformatics* **28**, i32–i39 (2012).
14. Coelho, L.P. *et al.* Determining the subcellular location of new proteins from microscope images using local features. *Bioinformatics* **29**, 2343–2349 (2013).
15. Chebira, A. *et al.* A multiresolution approach to automated classification of protein subcellular location images. *BMC Bioinformatics* **8**, 210 (2007).
16. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
17. Pärnamaa, T. & Parts, L. Accurate classification of protein subcellular localization from high-throughput microscopy images using deep learning. *G3 (Bethesda)* **7**, 1385–1392 (2017).
18. Kraus, O.Z., Ba, J.L. & Frey, B.J. Classifying and segmenting microscopy images with deep multiple instance learning. *Bioinformatics* **32**, i52–i59 (2016).
19. Nathalie Japkowicz, S.S. The class imbalance problem: A systematic study. *Intell. Data Anal.* **6**, 429–449 (2002).
20. Coelho, L.P., Peng, T. & Murphy, R.F. Quantifying the distribution of probes between subcellular locations using unsupervised pattern unmixing. *Bioinformatics* **26**, i7–i12 (2010).
21. Zhao, T., Velliste, M., Boland, M.V. & Murphy, R.F. Object type recognition for automated analysis of protein subcellular location. *IEEE Trans. Image Process.* **14**, 1351–1359 (2005).
22. Shen, Y.-Y.X.L.-X.Y.H.-B. Bioimage-based protein subcellular location prediction: a comprehensive review. *Front. Comput. Sci.* **12**, 26–39 (2018).
23. Khatib, F. *et al.* Algorithm discovery by protein folding game players. *Proc. Natl. Acad. Sci. USA* **108**, 18949–18953 (2011).
24. Khatib, F. *et al.* Crystal structure of a monomeric retroviral protease solved by protein folding game players. *Nat. Struct. Mol. Biol.* **18**, 1175–1177 (2011).
25. Chris, J. *et al.* Galaxy Zoo: 'Hanny's Voorwerp', a quasar light echo? *Mon. Not. R. Astron. Soc.* **399**, 129–140 (2009).
26. Clery, D. Galaxy evolution. Galaxy zoo volunteers share pain and glory of research. *Science* **333**, 173–175 (2011).
27. Raddick, M.J. *et al.* Galaxy Zoo: exploring the motivations of citizen science volunteers. *Astron. Educ. Rev.* **9**, 18 (2010).
28. Lee, J. *et al.* RNA design rules from a massive open laboratory. *Proc. Natl. Acad. Sci. USA* **111**, 2122–2127 (2014).
29. Sørensen, J.J. *et al.* Exploring the quantum speed limit with computer games. *Nature* **532**, 210–213 (2016).
30. Hughes, A. *et al.* Quantius: Generic, high-fidelity human annotation of scientific images at 10^5-clicks-per-hour. Preprint at https://www.biorxiv.org/content/early/2017/07/15/164087 (2017).
31. Danielle, N., Shapiro, J.C. & Mueller, P.A. Using mechanical turk to study clinical populations. *Clin. Pyschol. Sci.* **1**, 213–220 (2013).
32. Cox, J. *et al.* How is success defined and measured in online citizen science? A case study of Zooniverse projects. *Comput. Sci. Eng.* **17**, 28–41 (2015).
33. Feng, W., Brandt, D. & Shah, D. A long-term study of a popular MMORPG. *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games* 19–24 (2007).
34. Warfield, S.K., Zou, K.H. & Wells, W.M. Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation. *IEEE Trans. Med. Imaging* **23**, 903–921 (2004).
35. Snow, R., O'Connor, B., Jurafsky, D. & Ng, A. Cheap and fast, but is it good? Evaluating non-expert annotations for natural language tasks. *Conference on Empirical Methods in Natural Language Processing* 254–263 (2008).
36. Calise, S.J. *et al.* Glutamine deprivation initiates reversible assembly of mammalian rods and rings. *Cell. Mol. Life Sci.* **71**, 2963–2973 (2014).
37. Carcamo, W.C. *et al.* Induction of cytoplasmic rods and rings structures by inhibition of the CTP and GTP synthetic pathway in mammalian cells. *PLoS One* **6**, e29690 (2011).
38. Handfield, L.F., Chong, Y.T., Simmons, J., Andrews, B.J. & Moses, A.M. Unsupervised clustering of subcellular protein expression patterns in high-throughput microscopy images reveals protein complexes and functional relationships between proteins. *PLOS Comput. Biol.* **9**, e1003085 (2013).
39. Hasanpour, S., Rouhani, M., Fayyaz, M. & Sabokrou, M. Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures. Preprint at https://arxiv.org/abs/1608.06037 (2016).

ONLINE METHODS

Images from the HPA Cell Atlas. In this article, we are classifying protein distribution patterns from the publically available Human Protein Atlas (HPA), Cell Atlas database (https://www.proteinatlas.org). The HPA Cell Atlas project aims to characterize the subcellular distribution patterns of the entire human proteome using an antibody-based approach and confocal microscopy. Here, we have used the images and annotations from v14 of the HPA Cell Atlas, where proteins were classified into one or more of 20 organelles and cellular structures (in total 226,732 images of which 65,596 were public in v14).

Proteins are cataloged serially using in-house generated antibodies and immunostaining in a gene-centric manner as described in detail previously[7]. Briefly, the spatial distribution of each protein is studied in three cell lines out of a panel of 17; U-2 OS and two additional selected to have the highest RNA expression level of the corresponding gene. Each antibody-cell line 'sample' is then imaged to produce a minimum of two images per sample (average 2.93 images per sample). Each 'image' in the HPA Cell Atlas consists of four channels acquired sequentially with a Leica SP5 confocal microscope (DM6000CS) equipped with a 63× HCX PL APO 1.40 oil CS objective (Leica Microsystems). The settings for each image were as followed: Pinhole 1 Airy unit, 16bit acquisition and a pixel size of 0.08 µm. The detector gain measuring the signal of each antibody was adjusted to a maximum of 800 V to avoid strong background noise. A small part of the plates was imaged automatically using the MatrixScreener M3 in LAS AF software (Leica Microsystem). Here, z-stacks at six FOVs were acquired. False-colored channels represent the protein of interest (green), DAPI labeling of the nucleus (blue), microtubules (red), and the endoplasmic reticulum (yellow). Each channel is stored as a separate 2,048 × 2,048 16-bit ome-tiff.

Additional information on the experimental materials and reproducibility can be found in the Life Sciences Reporting Summary.

Tree structured annotations. Classes in the HPA Cell Atlas can be viewed as a tree structure, where depth in the tree increases, annotations become more specific. At its base, the cell is divided into the nuclear and cytoplasmic spaces. These two super-classes can be further divided into meta-classes. The nuclear super-class into; nucleus, subnuclear structures, nucleoli, and nuclear membrane. The cytoplasmic super-class into; cytoplasm, cytoskeleton, MTOC, secretory, and cell periphery. Lastly, these meta classes can be divided into the leaf node classes used in this publication. When discussing this structure in terms of PD, votes are first pooled and a hypergeometric test is performed to calculate consensus as described below at each level of the tree. As there are fewer options to choose from, nodes near the root of the tree require more evidence to be considered significant (hypergeometric test $P < 0.01$). When discussing this structure in terms of the DNN approach using the localization cellular annotation tool (Loc-CAT), each level of the tree represents a separately trained model.

Immunostaining after induction of R&R formation. U-2 OS cells were cultivated in McCoy's 5A modified medium (Sigma Aldrich) with 10% FBS and 1% L-glutamine (Sigma Aldrich), at 37 °C in a 5% CO_2 humidified environment. The cells were harvested at 60–70% confluency and seeded onto a glass bottom plate (Greiner Sensoplate Plus, Cat# 655892, Greiner Bio-One) coated with fibronectin (Sigma-Aldrich). 6 h before fixation Ribavirin was added to the growth medium to a final concentration of 0.15 mM. PBS-washed cells were fixed in 4% paraformaldehyde (PFA) in growth media supplemented with 10% FBS for 15 min, followed by permeabilization with 0.1% Triton X-100 in PBS for 3 × 5 min. After a washing step with PBS, cells were incubated with the primary antibody overnight at 4 °C. Rabbit polyclonal HPA antibodies were diluted to 2–4 µg/ml in blocking buffer (PBS with 4% FBS) containing the R&R marker (Abnova Corporation Cat#H00055466-M01, RRID:AB_426011) diluted to 1 µg/ml in blocking buffer. The next day, cells were washed 4 × 10 min with PBS followed by 90-min incubation at 20–22 °C with the following secondary antibodies (all from ThermoFisher Scientific) diluted to 1 µg/ml in blocking buffer: goat anti-rabbit AlexaFluor 488 (A11034, RRID: AB_2576217), goat anti-mouse AlexaFluor 555 (A21424, RRID:AB_2535845). Cells were finally counterstained with DAPI for 10 min, before being mounted in PBS containing 78% glycerol.

Ground truth for evaluation. The training labels for evaluating methods presented in this work are based on three rounds of manual curation performed for the HPA Cell Atlas v14 (**Supplementary Data Set 1**). Images were first annotated manually by a trained expert and labels were given based on all images in a sample. This was followed by a review in which stainings from multiple cell types were compared and consistency of staining was assessed. Lastly, a thorough literature review was performed. Annotations were corrected as needed throughout this process.

Expert reannotation. To assess consistency of labels in the HPA Cell Atlas, internal experts were presented with a random subset of 660 samples for reannotation from samples that were publicly available in the HPA Cell Atlas v14 (**Supplementary Data Set 3**). All reannotations and statistics measuring the accuracy of these reannotations were calculated at the per-sample (group of images) level. This gives an advantage to experts over the other methods in this work as some images in a sample do not contain the annotated label. For historical reasons, this reannotation did not include a distinction between Nucleus and Nucleoplasm, so the expert score is inflated slightly. Assuming the expert performance on the Nucleus-Nucleoplasm split was comparable to Loc-CAT and gamers, the expert per-class precision and recall would drop from 0.74 and 0.69 to ~0.70 and ~0.66 respectively, still well above all other methods. Microtubule ends were not present in the reannotation set as it is very rare. As a proxy, average performance was assumed for this class. This may be generous considering both gamers and Loc-CAT struggled with this class. In the worst case, were experts to entirely miss this class, per-class precision and recall would drop to ~0.69 (~0.67 with nucleoplasm) and 0.63 (~0.62 with nucleoplasm).

Statistics and reproducibility. In this work we used several metrics to measure the performance of both PD participants and the Loc-CAT DNNs, hence forth referred to as 'predictors'.

Assessing performance. To assess the agreement between generated predictor annotations and HPA Cell Atlas v14 annotations, we assessed precision and recall as defined in equations (1) and (2) below.

$$Precision = \frac{TP}{(TP + FP)} \tag{1}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{2}$$

Here, true positives (TP) are annotations for which the predicted label matches the prior HPA label, false positives (FP) are predicted labels that the HPA has not identified, and false negatives (FN) are labels that the HPA had annotated which the predictor did not predict. Again, note that in cases of labels which are novel to the HPA, such as nucleoli (rim), the FP = 1 and FN = 0 by definition as the HPA has never previously annotated this localization.

In the case of multi-label data, accuracy cannot directly be assessed, as label confusion cannot be readily defined. To measure per-class performance we used F1 score which is the harmonic mean between precision and recall.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

Measuring cross-localization confusion. Due to the multi-label nature of the problem, it is impossible to construct a confusion matrix indicating what labels predictors select in comparison to those annotated in the HPA Cell Atlas v14. In an attempt to understand confusion, we compute a matrix indicating the probability that the frequency of specific multi-localizations occurs based on HPA Cell Atlas v14 colocalization probabilities. In doing this, we compare the probability of observing location B_{HPA} given location A_{HPA} as defined by the HPA Cell Atlas v14 annotations ($P(B_{\mathrm{HPA}}|A_{\mathrm{HPA}})$) with the frequency of prediction for the localization $\hat{B}$ given A_{HPA} using a one-tailed binomial test as indicated in equation (4) below. Note that this test only measures over co-annotation.

$$P_{\mathrm{binomial}}\left(F(\hat{B}\,|\,A_{HPA})\,|F(\hat{B}), P(B_{HPA}\,|\,A_{HPA})\right) \tag{4}$$

Where $F(\hat{B}|A_{HPA})$ is the frequency of predicted location $\hat{B}$ given an observed label A_{HPA}, and $F(\hat{B})$ is the number of all predicted localizations $\hat{B}$ independent of corresponding HPA annotation. The test is used to measure over co-annotation in the multilabel case, and will be significant if the predictor annotates one category significantly more frequently than we expect given

the co-annotation probabilities by the HPA Cell Atlas v14. This can occur either via confusion, where one label is incorrectly identified as another with regularity, or general over-annotation where a predictor is biased to more frequent annotation of a given label. Note that the test is never significant on the diagonal where $P(B_{HPA}|A_{HPA}) = 1$. The resulting p-values were then subjected to a Bonferroni multiple hypothesis correction per-class ($n = 29$). These results are presented as a circular plot serving as a proxy for multi-label confusion (**Figs. 3** and **6**). As over-annotations are directional, tapering is used to indicate directional confusion, with the thick end of the ribbon indicating which class is over-annotated by the predictor (confused with) together with the HPA Cell Atlas v14. Ribbons with two thick ends indicate a bi-directional over-representation of the co-annotation. This can also be viewed in tabular form (**Supplementary Figs. 3** and **5**).

Reproducibility. All images in the HPA Cell Atlas v14 ($n = 226,732$ images of which 65,596 were public in v14) consist of 4 false-colored channels as shown in **Figure 1a**. The number of images containing each of the patterns in the **Figure 1b–d** can be found in **Supplementary Figure 3**. The authors note that this is not the number of images containing only this pattern as multi-localization of proteins causes more than one pattern per image. All 10,003 protein coding genes publicly available in HPA Cell Atlas v14 are assayed in three cell types; U-2 OS and two selected based on maximal RNA expression (FPKM/TPM) creating >10,003 such morphological variability replicates in HPA Cell Atlas v14 such as the examples seen in **Figure 1e**. The numbers for each specific cell type can be seen in **Figure 5c**. Of the proteins analyzed in this study, 44% ($n = 101,903$) of images (not proteins) contain multi-localizing proteins such as those shown in **Figure 1f**. Cell-to-cell variability (**Fig. 1g**) was a new category in v14 and therefore contained no true-positive images. In the updated Cell Atlas v16 containing this cell-to-cell variability analysis 1,896 protein coding genes (most with 6+ images per protein coding gene) are annotated as having variable patterns.

Players in PD contributed ~33 million image annotations. Annotations for each image are pooled into a consensus using a hypergeometric test (minimum 12 votes, see Online Methods). Results of these consensus annotations each day are compared with gold standard HPA Cell Atlas v14 to obtain an F1 score per day which demonstrates a stable behavior after 100 d (**Fig. 2d**).

Comparing the overall F1 score of players ($n = 59,901$) who have analyzed a minimum of ten images, with a consensus built on hypergeometric tests based on the cumulative consensus (pooling individual votes from all rounds, median 78 votes per image) demonstrates the power of pooling multiple votes (**Fig. 3a**). Over-represented co-annotations are measured using a set of pair-wise binomial tests where the null hypothesis is the expected co-localization probability in the HPA Cell Atlas (**Fig. 3b**). Replicate numbers of images containing proteins annotated to each class under both the HPA Cell Atlas, and PD can be found in **Supplementary Figure 3**. There were 1,498 images annotated centrosome and 424 images annotated MTOC in the HPA Cell Atlas (**Fig. 3c**).

Histograms showing the counts of images annotated for each class are shown in **Figure 4a**. These numbers are either directly counted from the HPA Cell Atlas v14 data set (gold), or based on the 65,596 public images in the HPA Cell Atlas v14, where a hypergeometric test is performed on the pooled annotations for each image (median $n = 78$ annotations per image). Performances (F1 score) in the tree based hierarchy (**Fig. 4b**) are based on hypergeometric consensus for each image. The number of class instances in HPA Cell Atlas v14 can be found in **Supplementary Figure 3**, rows labels. Example images of Rods & Rings proteins are shown based on the ten proteins discovered by players of PD. Independent colocalization experiments under ribavirin induction with a marker for R&R for each protein were performed in triplicate (see Online Methods).

Performances in the tree based hierarchy (**Fig. 5a**) are based on the number of class instances in HPA Cell Atlas v14 (**Supplementary Fig. 5**). Example images (**Fig. 5b**) are the worst-case picked from a rank-list of each of the lowest performing classes from the hierarchical tree (**Fig. 5a**). These images are meant for illustrative purposes of the types of mistakes Loc-CAT makes in the worst case. Performance on each cell line (**Fig. 5c**) and compared across single and multi-label data (**Fig. 5d**) is based on the average of fivefold cross validation.

Overlap in Loc-CAT predictions and PD predictions (**Fig. 6a**) are based on the number of class instances in HPA Cell Atlas v14 (**Supplementary Fig. 5**).

Over-represented co-annotations are measured using a set of pair-wise binomial tests where the null hypothesis is the expected co-localization probability in the HPA Cell Atlas (**Fig. 6b**). Replicate numbers of images containing proteins annotated to each class under both the HPA Cell Atlas, and Project Discovery can be found in **Supplementary Figure 5**. Individual player performance (**Fig. 6c**), compared to consensus performance of Project Discovery, Loc-CAT performance using various training architectures (Loc-CAT, Loc-CAT+, GA Loc-CAT), and expert annotations are based on the 65,596 images in the HPA Cell Atlas v14. Scores are computed per-class, where true-positive instances of each class can be seen in **Supplementary Figure 5**.

Experimental reproducibility. Additional information on the experimental materials and reproducibility can be found in the Life Sciences Reporting Summary.

PD: MMO science. This work presented a new approach to citizen science and gamification. Termed Project Discovery, this method is the first to utilize main-stream MMO games to perform real scientific research. This effort was a collaboration with CCP Games (EVE Online) and MMOS.

Image preparation. Images were converted from 2,048 × 2,048 16-bit greyscale tiff images to RGB false color 1,200 × 1,200 jpeg images with 89% compression. The resulting images were then given randomized names and uploaded to MMOS Amazon Web Servers. This configuration was chosen to limit server load as each color channel could be directly dropped on the EVE client side after the image was served. For this reason, the players did not receive a color channel for the endoplasmic reticulum (ER). This also limited the ability of colorblind players to participate, though we suggested that such players use a 'shader' to shift the screen into a colorblind-friendly palate.

For each batch of images, a tab separated plain text metadata file was generated and uploaded to an MMOS Amazon Web Server. Each row of a metadata file represented one image in the batch and the columns of each metadata file were used to provide information about the image. In addition, a json formatted 'control' file was generated for each batch specifying information about the batch including the number of images, and version number.

Game play. The mini-game within the EVE Online universe was accessible from anywhere in-game allowing maximum access for players. The game design was created by CCP games and students at Reykjavik University.

In the game, players were presented a false-color confocal microscopy image and are tasked with classifying the green pattern into up to 5 of the 29 predefined categories. Players could use the blue (nucleus) and red (microtubules) channels to assist them and could toggle these color channels on and off as well as zoom in on the image by hovering. Players could compare the patterns seen in each image with five reference images of each pattern visible upon hovering over each tool-tip in-game. These images were carefully selected to represent the diversity of the respective staining patterns across the multitude of cell lines.

After submitting a classification, players received an in-game reward in the form of in-game currencies that could be used to purchase in-game items exclusive to PD as well as level-badges. Players received one small reward per-sample analyzed, plus a larger reward for each time they leveled up. Initially players also received a bonus reward based on their agreement with the eventual community consensus, however this was quickly exploited with players converging on a single common class (Cytoplasm) and this reward was therefore discarded. Players were also provided with a 'pass' option after expressing that some images were too challenging and they would rather pass than make a bad guess.

In an additional attempt to control accuracy, control samples in which the solutions were known a priori were provided at random intervals. If player performance drops too low, the player is returned to the tutorial phase.

To view a tutorial of game play, please visit our youtube channel (https://www.youtube.com/channel/UCfUA1LRafjldAom5lzSQD7A/videos?view_as=subscriber).

Tutorial and training. To control data quality, players were required to complete a tutorial and training phase before contributing to classifications of unknown samples in PD. Players were entered into the tutorial and first asked to classify images of easy, single-localizing protein to a restricted set of localizations to familiarize themselves with the user interface. Once past the tutorial, players entered training, where players were presented with increasingly difficult

doi:10.1038/nbt.4225

samples and were required to correctly annotate these before passing the training phase and being allowed to contribute annotations for unknown samples to the project. Player accuracy was measured with random control samples which were identical to test samples but had been pre-annotated by experts from the HPA Cell Atlas. If player performance dropped below a threshold, players were returned to the training phase of the game until their performance improved to a level that they were allowed to contribute to the consensus again.

Consensus calculation. Tasks were presented to gamers in a randomized order. To control for erroneous annotations, we asked multiple gamers to annotate each image. A minimum of twelve gamers assessed each image before it could be evaluated for consensus. We measured 'consensus' on a task using a hypergeometric test as described in equation (5) below assuming that each player chose the maximum of five classes per task. This test assumes each class is independent and as such it does not account for mutual exclusivity of tasks (for example, Nucleoplasm with Nucleoli). The test is given by

$$P = 1 - \sum_{i=1}^{m} \frac{\binom{p}{i}\binom{N*n-n}{m*n-i}}{\binom{N*n}{m*n}} = 1 - \sum_{i=1}^{5} \frac{\binom{n}{i}\binom{28*n}{5*n-i}}{\binom{29*n}{5*n}} \quad (5)$$

where, n is the number of players that have voted on the task, N is the number of classes available (29), m is the maximum number of allowed classes per sample (5). This equation gives an estimate of the probability that each category had been selected m times given n tries (gamers). Once 12 votes were acquired, this CDF was evaluated after each subsequent vote. If the likelihood of at least one category was statistically significant ($P < 0.01$), and no other categories were near the significance boundary ($0.01 < P < 0.1$) we considered a consensus reached and the task was closed. The hypergeometric test measures the probability of obtaining k 'hits' in n random draws from a set without replacement. This test is also extremely efficient to compute, making it feasible for the real-time computation with high server loads experienced of over 800 submissions per minute.

After six rounds of annotations, votes from each round were aggregated and the consensus recalculated (average 97 votes per image). As statistical significance was increased due to the increased number of samples, this created a more sensitive test for rare and under annotated classes, however it also exacerbated the over-annotation problem for common classes. To correct for this effect p-value cutoffs were tuned per class on a held out 10% of the data of the based on the expected class distribution from the previously annotated HPA data. Novel classes were set to the highest allowable p-value cutoff (0.01) for discovery.

When constructing consensuses for meta classes, votes were merged into super categories, and then re-evaluated using the hypergeometric test and the aforementioned procedure given the presence of fewer classes.

Expectation maximization. Jointly estimating individual player bias together with the true label can be done via expectation maximization (EM). In this work, we implemented a binary EM for each class based on the STAPLE method[39], as multi-label data makes direct multi-class evaluation impossible. Due to the computational time required, we ran the algorithm for 10–30% of the data set (n = 6,558–19,534) respectively. We observed no improvement when increasing the percentage of the data set evaluated and report the best accuracy of all runs (**Supplementary Table 1**). Unfortunately, as the number of single labels is very high (29), the frequency of most labels is very low (<0.1%), and the number of images analyzed per player is relatively low (mean = 44), this method did not improve results and the previously discussed consensus calculation was used instead.

Project appeal. Project appeal (**Fig. 2b**) was calculated as defined in[32] and given in equation (6) below.

$$Project\ Appeal = \frac{Number\ of\ volunteers}{(Project\ active\ period)^2} \quad (6)$$

Loc-CAT: DNN protein localization. This work presented a feature based multi-label DNN model for predicting subcellular protein localization. This network outputs a real-valued confidence vector with a score for each possible class.

Image feature extraction. Quantitative image features were calculated using MATLAB 2016a. Image processing was performed at a per-cell level on each image consisting of four fluorescent microscopy images, one for each acquisition channel. The DAPI images were first treated with a low pass filter followed by active contour segmentation. Cells were then segmented using a combination of the microtubule and ER channels and seeded watershed. Cells with nuclei touching the image edge are removed from classification, though the cytoplasm of the cell can contact the edge of the image.

After segmentation, a set of 2,233 quantitative SLF image features were extracted based on work by the Murphy Lab[9,20]. Of these, the 719 features describing the green fluorescent channel in relation to the other channels, were passed to Loc-CAT (https://github.com/CellProfiling/Loc-CAT). These features describe the intensity, texture, and spatial relationships between the protein of interest (green) and remaining fluorescent channels of the image. The remaining unused features describe the relationship of reference channels to themselves and are used internally in other applications. Details on each feature can be found at (http://murphylab.web.cmu.edu/services/SLF/features.html, **Supplementary Data Set 2**).

Data partitioning. Images are shuffled at the sample level, meaning though images of the same antibody in another cell line may be present in the test set (for cross-cell line classification), all images of an antibody in a specific cell type will be in a single fold. Images are then partitioned into five folds by sample. Each training set contains 80% and each testing set contains 20% of the available data. Training sets are then split again by sample into training (90% of training set) and validation (10% of training set) sets. The resulting training set was then shuffled per-cell to avoid bias in the training. Because folds were shuffled created per-sample, it is possible that each fold contains variable number of images and cells. After data partitioning, input features to Loc-CAT are Z-normalized on the training set (excluding the validation subset).

Neural network architecture. In Loc-CAT, CPython 3.5.2 with CUDA gpu-accelerated TensorFlow (v1.3.0) was used to train a feed-forward deep artificial neural network containing three hidden ReLU6 layers with 800 neurons per layer and a sigmoid output function. Dropout was applied to reduce the risk of overfitting and better generalize the network, 20% on the input layer and 40% on each hidden layer. The network was optimized using the ADAM optimizer and a binary-cross entropy loss function. In the development of Loc-CAT, several network architectures were tested using a multidimensional parameter sweep (trained and tested on U-2 OS images from HPA Cell Atlas v.14).

Stopping rule: during training, if the cost on the held-out validation set did not decrease for ten epochs, training was halted. The network weights were then reset to the epoch before those ten epochs.

Prediction aggregation. As the quantitative image features are extracted per cell, the classifier predicts localization for individual cells rather than images. When training the network, binary cross entropy was applied to these per-cell annotations, using the HPA annotation for the image the cells came from as the true label. Location predictions were aggregated for all cells from the same image by taking the mean predicted value for each class. The cutoffs for each class are then tuned at the IMAGE level for the first fold of the testing set to optimize the per-class performance. Average performance for each of the remaining four test-set accuracies are reported using these cutoffs.

All single-cell line classifiers reported the average statistics of fivefold cross validation. Cross-cell line classification statistics are based on predictions for all samples in the testing cell type and therefore cross validation does not apply. In the hierarchical tree (**Fig. 5**), each level of the tree was trained separately and tested using fivefold cross validation.

Gamer-augmented transfer learning. The gamer transfer learning network was trained using the same network structure as before with the P values calculated from the gamers' consensus added concatenated to the input features (**Supplementary Fig. 4**).

For the pseudo-gamer transfer learning network (Loc-CAT+), a secondary network was trained to predict the gamer consensus P values (**Supplementary Fig. 4**). The secondary network was trained for 100 epochs on the same SLF input features with two hidden ReLU6 layers containing 200 and 100 neurons, respectively. Dropout was applied to the secondary network as well, 20% on the input layer and 40% on each of the hidden layers. The predicted P values are then concatenated to the standard SLF features as input to the standard Loc-CAT network.

CNN. We evaluated a convolutional neural network using the SimpleNet architecture with dropout[39]. The network was trained using versions of HPA images scaled to 128 × 128 pixels as input. Each input image contained all four available image channels. The network was trained for 600 epochs with no substantial validation loss change seen for the last 200 of those epochs. The performance, although inferior to the other methods presented in this paper,

showed great promise for a convolutional network properly trained and tuned for protein localization.

Protein of interest only classifier. Loc-CAT architecture was trained using only features from the protein of interest; however, performance of this classifier was substantially inferior to that of the model trained with the three cellular reference channels (data not shown). This suggests that the contextualization of the protein in a cell using such reference markers is crucial for accurate protein localization from images.

Life Sciences Reporting Summary. Further information on experimental design is available in the Nature Research Reporting Summary linked to this article.

Data availability statement. The images included in this study are available in the HPA Cell Atlas (https://www.proteinatlas.org), specifically the HPA Cell Atlas v14 can be found at (https://v14.proteinatlas.org). The data from Project Discovery is available upon request.

Code availability statement. Code for extracting features from images in the HPA Cell Atlas is available at: https://github.com/CellProfiling/FeatureExtraction. Code for the analysis of data from Project Discovery presented in this work is available at: https://github.com/CellProfiling/ProjectDiscovery. Code for the Loc-CAT presented in this publication is available at: https://github.com/CellProfiling/Loc-CAT.

doi:10.1038/nbt.4225

3.8. **Rationale for Evidence #4 Paper**

This paper really allowed me to understand the Proteins dataset and see what is wrong with my own networks. The paper was written before the Kaggle challenge was put up and it details two different methods that were used to classify the Human Protein Atlas dataset. One of the methods they used was to implement a mini-game in which players do the classifications for them. This was done with the help of the makers of EVE Online, a popular multiplayer online game, who made the classification task a mini-game within their popular game. In order to get people to play this game, the makers incentivized players with in-game currency. While the players were able to classify many images, the researchers realized that the game could not go on forever, as the number of players would decrease over time, and the game company could not keep the game running forever. This fact is why they decided to develop a neural network to classify these images better. After training their model, the researchers were able to make a network that had a final F1 score of 0.71(which was greater than the top person of the Kaggle leaderboard).

Even though their model was radically different than both of my models, the paper showed me what I did wrong, and allowed me to answer two of my guiding questions. In order to justify their methods, the paper describes the many uses that can come out of classifying these proteins. The researchers say how these classifications can help researchers analyze the images faster as the entire process of classification takes a lot of time. Speeding up this process would allow researchers to understand the proteins function faster and could even be used to understand diseases. This intro opened up my eyes to the possible uses of my networks in the real world. The paper also showed me what I was doing wrong when implementing my models. While reading about how they made Loc-CAT, the researchers

said that they used a sigmoid activation function for their last layer and the Adam optimizer for their network. I realized that these two things were what could fix both the CNN and the CNNLSTM. After implementing these two things, all of the models' values were fixed and I had two workable models.

4. Journal Entries

4.1. Reflection Journal #2

Senior Focus Field Work Reflection Journal

FLOW PROBLEMS

May 17, 2019

Tanay Kane

This week was actually very productive. My goal for the week was just to walk through a full competition on Kaggle and how I would load my data in and how I would process it. Although in the end, I was able to reach my goal, I had run into many problems that I need to talk to my mentor about because I have no idea why these problems occurred.

The biggest problem that happened was with loading the images into the program. My original plan was to use the `flow_from_dataframe()` function in keras. The `flow_from_dataframe()` function takes in a directory and a data frame as an input and is able to organize what images belong to what classes. The, in this case, would be a chart that would contain all of the image names in one column and in the other column what the possible class(es) the image belonged to. I wanted to really use this function over the `flow_from_directory()` function, that I had to use because it would be more helpful regarding the proteins images. Since each of the images of cells can show multiple things. For example, one of the images contains both cells that have a Cytokinetic Bridge and the nucleoplasm of many of the cells. This would be a pain to do with the `flow_from_directory()` function as I would have to put this image in two different directories and eat up my computer's memory. The `flow_from_dataframe()` function would have allowed me to just feed in a chart of all the image names and what their labels are.

However, when I tried to implement `flow_from_dataframe()` while doing the Cancer Cell detection I kept getting either Found 0 images belonging to 0 classes, or some kind of error saying that the data frame doesn't work. I was absolutely baffled and could not figure out why it wouldn't work. I changed my paths, made the paths more explicit, and even made the data frame more explicit, but in the end, I could not figure out why. I had to settle for the `flow_from_directory()` function instead.

This really bothered me as I do not want to use this function when it comes time to work on the proteins dataset. I will have to look into it more next time and talk with my mentor to see why it did not work out too well. However, I am still happy with what I got out of it. I was able to learn what I would have to do to work with my dataset and how I would submit my data to Kaggle for evaluation. Even though I got a very low accuracy rate, I was pleased with the progress I made.

4.2. **Rationale for Reflection Journal #2**

I chose this reflection journal because it really shows my mindset going into this project. At the time, I knew nothing, and my only goal was to be able to make CNN and CNNLSTM and train it on the Human Protein Atlas dataset. In order to do this, I knew that I could not jump in head first and needed to familiarize myself with how I would even go about it. Because of this, I decided to run through a sample Kaggle challenge (the dataset was from Kaggle) first and then develop the CNN and CNNLSTM. Looking through the list of challenges, I found the Histopathologic Cancer Detection Challenge. With my little understanding of biology, I thought that the two datasets were similar enough, so working with this dataset would surely allow me to have the skills to work with the Proteins dataset. However, looking back at it, this was very naive as the Proteins dataset was way different than this one. For example, this dataset was binary, meaning that each of the labels for this dataset was either "Cancer" or "No Cancer" whereas the Proteins dataset has 28 possible labels and different combinations of each of those labels. Not to mention that the types of cells were completely different.

This journal also shows some of the beginning frustrations that I had with Keras. While preprocessing the dataset, my problems with the `flow_from_dataframe`, started to scare me as there were too many possible things that were going wrong. However, I was able to overcome these problems by just using the `flow_from_directory` method instead. Even though I got over this problem, I was still scared as I did not want to use the same method for the Proteins dataset as I would need 28 separate folders for each of these classes and because each image has multiple classes, I would have multiple versions of each of the images, which would take up a lot of space. However, for the time being, I had overcome this little obstacle

and was able to run all of the images through a simple CNN and get my results. Even though the model was not very accurate and my choice of the dataset was a bit uneducated, I chose this journal because that week had taught me a lot of things. Without the work that I did during this week, I would not be as familiar with Keras as I am now and would not know how to go through with preprocessing any of the images that I would later use throughout the course. Knowing what I know now, I am glad that I didn't start off by jumping into the Proteins dataset head-on with the limited knowledge that I had. Instead, by taking the time to learn Keras and walk through one of the other challenges, I was able to get a feel for what I would be dealing with in the future and knew how to go about it.

4.3. **Reflection Journal #5**

Senior Focus Field Work Reflection Journal

A ROUGH WEEK

May 17, 2019

Tanay Kane

This week was kind of disappointing in terms of focus work. Last week I was really optimistic as I saw an intersection between my focus work and my robotics work and I really wanted to pursue it hoping that I could benefit both of them. Instead, I was left with nothing and in turn, had to just abandon it altogether. It was a rough week as I had to really look at all of the time that I had spent on it and ask myself whether or not this would be ready in time. To add on to that, I had lost the robotics competition altogether.

Looking back on it, I think I was too quick to jump onto this opportunity in the first place. It would have been great to gather this data as I could have talked about the importance of the accuracy vs speed tradeoff that happens when these neural networks are implemented into the real world. I was even really excited as I had a live app that could show off my model, identification in real time. However, all that happened came out of this endeavor was just wasted focus time.

However, after the robotics competition, I was really able to step back and look at the focus work that I have ahead of me and the work that I have done. A lot of the time that I have spent on focus has been stalled by problems that had easy solutions, but that was hard to find. The best example is the `flow_from_dataframe()` vs. `flow_from_directory()` problem that occurred during my second week (?) of fieldwork. I knew that the `flow_from_dataframe()` was the function I needed for the proteins dataset, but I spent too much time just trying to debug it and see why it was not working. In the end, I had found a tutorial where the person showed the `flow_from_directory()` function and it worked for me. This problem's solution could have taken a day to fix, but it really dragged on as I kept running in circles trying to find an answer. This same problem happened again this week as I was trying to find a way to convert the .ckpt file to the .pb file. Had I actually gone through and read the entire documentation, I would not have had to spend so much time trying to bash out a program that could do this.

The week was really eye-opening in terms of what I need to do to really finish the focus program strong. All I need to do is to really slow down and think things through more carefully. There are a lot of resources out there to help me out and I have a mentor who can answer all of the questions that I have about this. Looking into the coming weeks, all I want to do is to at least start training one of my models on the proteins dataset. I have the foundation to do it and just need a bit of research into the other CNN architectures to make the model and answer one of my guiding questions.

4.4. **Rationale for Reflection Journal #5**

The opening sentence is why I chose this progress journal. During this week I was really hoping that my robotics side project could allow me to learn more about CNNs in the real world. This side project dealt with differentiating between white balls and yellow cubes. The goal of the competition was to use preset computer vision algorithms to look at three objects and find the yellow cube. Once that yellow cube is found, you just have to move it over and continue with the rest of the competition. Throughout the season, I saw that their algorithm was a bit faulty and would slip up over the smallest thing. As a result, we would end up losing a lot of competitions. With these faults, I saw an opportunity to merge my focus time and my robotics time, so that I could have something for both the robotics competition and my focus presentation. The side project dealt with making and training a CNN to actually differentiate between the two game pieces and implement it something that would not be too taxing on what it was running on. Since the program would run on a Samsung, I could not have a massive file that required a lot of computing power to run. I did my research, found what I needed to do and started to train the model. However, even with everything that I did, this week was a real low in the entire focus period as I was left with nothing.

Despite having nothing, this reflection journal really shows a big change. With this big loss, I was able to reflect on what I would need to do going forward. Out of all of my reflection journals, I think this one was the most insightful overall. This journal is the only reflection journal where I was able to truly reflect on everything that I had done, and start to change the way that I would approach everything going forward. With everything not working, I was able to just take some time and see what I had to do going forward. Looking back at

it, this reflection journal allowed me to find the momentum to continue with everything and lead me to where I am now. In a way, if I had not lost the robotics competition and this side project had not failed, I probably would be making the same mistakes as I was making at the beginning. That is not to say that everything went fine after this week, but I was able to deal with my problems by thinking through them rather than brute force.

4.5. **Progress Journal #11**

DATA GATHERING WEEK

May 17, 2019

Tanay Kane

To my surprise, I actually did a lot more work than I expected. Last week, I said that all I wanted to do is to train both models on the FMNIST and MNIST datasets and if I had time, start the multi labeled dataset. Even though this week was a filled with a few lows, I was still able to get a lot done.

The first thing that I did this week was focus on the MNIST dataset for both the CNN and the CNNLSTM. MNIST stands for the Modified National Institute of Standards and Technology and the dataset contains many handwritten numbers. This dataset is usually used to introduce people to machine learning and image classification. I even started with this dataset when I began to do field work. Using the MNIST dataset was me starting at square one as I was familar with how each of the model should work and I could tell whether or not I had written a bad model or not. I first started with training the CNNLSTM on the dataset. For both models, I kept the epochs at 15 and set the `steps_per_epoch` at the number of total samples divided by 32 as this is standard for that value. Figure 1 shows the results from training and validating the dataset.

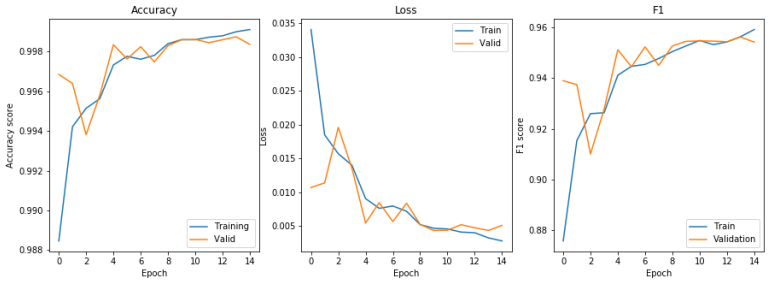

FIGURE 1. These are the loss, f1 score and accuracy score of the CNN LSTM over the 15 epochs on the MNIST dataset

Figure 1 shows that the model was very good as it followed the trends that it was supposed to. If you look at the accuracy and F1 graphs, you can see that the values increase very rapidly and then starts to flatten out. This trend means that over time, the model is getting better over time. The value start to flatten out at the end as the accuracy value and F1 value are getting close to 1, which is a perfect score on both scales. The loss was also great as it decreased over time. These values actually brought some relief to me as it meant that the CNNLSTM was good the way that it was.

After training the CNNLSTM, I moved on to the CNN. I kept everythin the same and started to train the model. However, when I generated the graphs, I was confused when I saw Figure 2. As Figure 2 shows, the model did not improve at all during training or

not seen. After running each of the models on the training set, I was actually a bit surprised that the two models performed similar to each other. Out of all of the images in the testing set, the CNN-LSTM had 3 more correct classifications than the CNN. I was a bit in a rush, so I did not compute the confusion matrix at the time, but once I have those data points, I can start to develop an answer to my topic question.

Even the MNIST data did give a lot of information on the two networks, it was only one multiclass dataset. If I truly wanted to know what network is better for multiclass classification, I would need another one. I decided that the next dataset that I would test both networks on was the FMNIST dataset. The FMNIST dataset contains images of different articles of clothing and was meant to replace the MNIST dataset. Everything in this dataset could have 1 of 10 things: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, or Ankle boot. This was another dataset that I had worked with previously, so I was a bit familiar with how it worked. Since this required little preprocessing, I was able to train both the CNN and CNNLSTM on this dataset fairly quickly. For training, I kept the

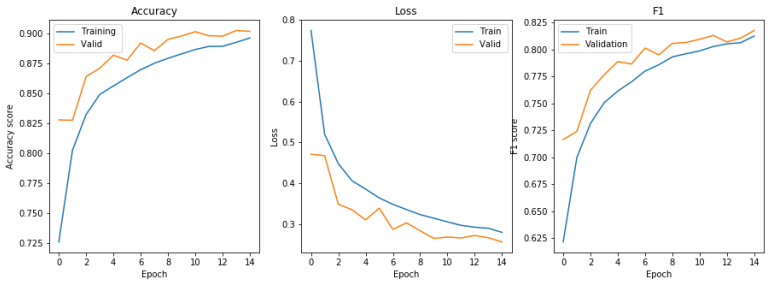

FIGURE 4. These are the loss, f1 score and accuracy score of the CNN for the FMNIST dataset

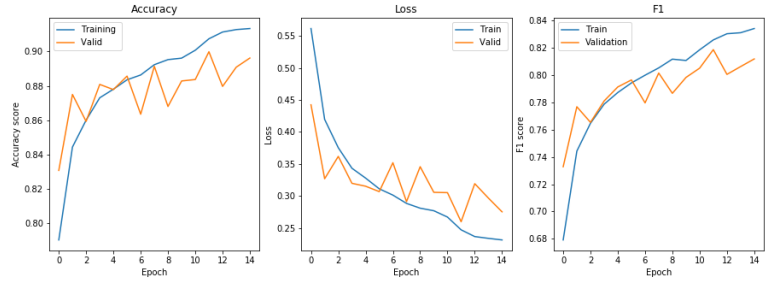

FIGURE 5. These are the loss, f1 score and accuracy score of the CNNLSTM while training it on the FMNIST dataset.

Figure 4 shows the results from the CNN's training and Figure 5 show the results from the CNNLSTM's training. As both Figures show, both models actually improved over time as their accuracy and F1 scores increased over time and their losses decreased over time.

After training these models it was time to test them. This testing is where I was a bit surprised. Out of the total number of testing images, the CNN actually labeled more images correctly than the CNN-LSTM, 67 to be exact. This means that the CNN was actually more accurate than the CNN-LSTM. However, I don't want to draw any conclusions yet as I did not calculate the confusion matrix for these results either.

The last thing that I did this week was to revist multi label classification. Multi label datasets are datasets where one image could have multiple classifications. The best example is the Proteins dataset that I started off with, since each of the images contained multiple things that helped build proteins. I wanted to see how the CNN and the CNNLSTM did on a less complex dataset, but that was still multi label. This would allow me to see how each network does on a dataset that is similar to the Proteins dataset and allow me to see why my data was so weird. In order to do this, I followed a tutorial (https://medium.com/@vijayabhaskar96/multi-label-image-classification-tutorial-with-keras-imagedatagenerator-cd541f8eaf24) that I had found online and adapted it to work with my model.

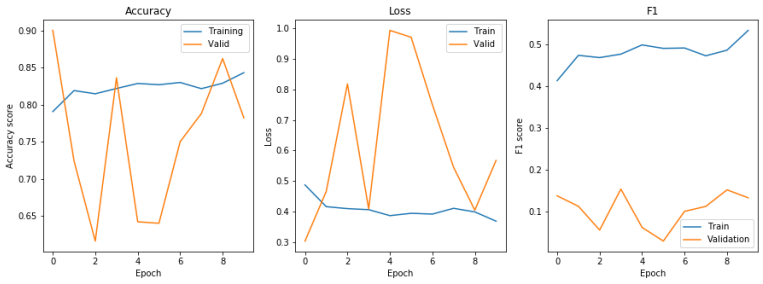

FIGURE 6. These are the loss, f1 score and accuracy score of the CNNLSTM during 10 epochs of training.

I followed the tutorial on how to preprocess and load my images into my models and actually trained my CNN and my CNN-LSTM on the dataset. For the training, the CNN-LSTM did not give me any problems and the trends, as shown in Figure 6, were as they should be. However the CNN was a bit different. Unfortunately, I did not save the graph, but what it showed was that the model was having a lot of trouble classifying images. In fact it kept defaulting to only one tag per image. Fortunately, the tutorial actually gave me a good suggestion. According to the tutorial, instead of ending on one dense layer with the number of neurons being the number of possible classifications, you could have a bunch of dense layers that correspond to one class. After rewriting the CNN to be this way, I retrained it and saw how it did over time.

Figure 7 displays how all of the values changed over time. Although both the accuracy and f1 score graphs are a bit hectic, all you need to know is that each line corresponds to a different class. Where as the other figures only had training and testing, Figure 7 has the training and validation values for each class. This allowed the model to label each of the images with more than one class. With these results, I feel that I should retrain the CNN-LSTM with a similar structure and see if that helps the CNN-LSTM make better decisions.

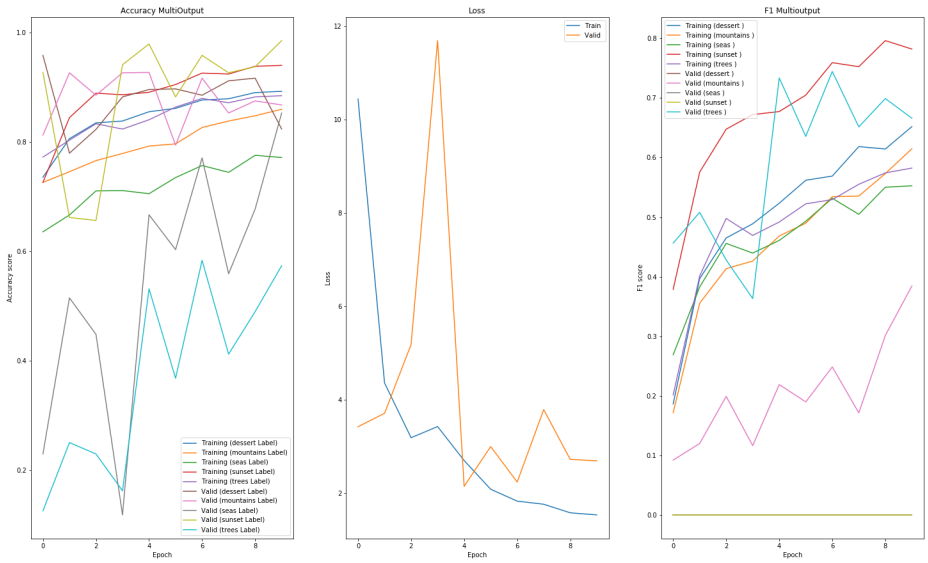

FIGURE 7. This graph shows the accuracy and f1 score of each of the CNNs outputs and the loss of the network as a whole.

4.6. **Rationale for Progress Journal #11**

This journal really laid out one of the ways that I would overcome an obstacle that occurred. When I got to the point where I could train both models on the Proteins dataset, my graphs came out very weird. For the CNN, the loss over time was very high and the accuracy and f1 scores were very low. On top of that, none of the trends made sense as the loss would increase and then end to where it started and both the f1 and accuracy scores were very bumpy, but would also end up at the same value that it started at. The CNNLSTM had not fared any better as most of its graphs were flat and would stay at the same value. Those values were also similar to the CNN's, so it wasn't much use. With both of the models having wacky trends and values, there were too many possible problems that could have caused this to happen. Instead of trying to try all of the possible solutions, I decided to shift the direction of my project to better suit the actual topic question. Originally, my plan was to just train both models on the Proteins dataset and use that data to make say whether the CNNLSTM was better than the CNN. However, with this weird data, I realized that I would need a lot more datasets to really reach a definitive conclusion. After a bit of thinking, I decided to train both networks on the MNIST and FMNIST datasets as I was very familiar with them. Seeing how both models do on each of the datasets would allow me to pinpoint what problems were occurring with each of the networks, and fix them. With these fixes, I would be able to tackle the proteins dataset again. Looking back, I can see how all of the upsets that occurred during the fifth week allowed me to get to this conclusion and really think through the problem. I came up with a solution that would actually help me reach an answer to the question that I was trying to answer in the first place.

4.7. Progress Journal #12

GOING BACK TO PROTEINS

May 6, 2019

Tanay Kane

In the last progress journal, I said that I was staring at two cross roads as I could either continue with the MNIST, FMNIST, and MIML datasets and try different CNN bases and modified versions of the CNNLSTM or I could use everything that I had learned with those datasets and try to go back to the proteins dataset one more time, and see if I can do better. Well, as the title says, I decided that I would just go back to the Proteins dataset and give it one more go. After doing my work this week, I actually think that I could have a workable model by the end.

The decision to go back to the proteins data was actually inspired by a paper that Ms. Stott gave me. In the paper, the authors discussed two ways that they tackled the protein classification problem and how the dataset was made [1]. The first method the authors used to help classify the proteins dataset was to set up a mini game in the popular massive multiplayer online game, EVE Online. This method was actually very helpful, but the authors knew that the mini game's popularity would not last long and the game company could not keep it running forever, so they decided to create a deep learning model that could classify the proteins. The system the designed was named Loc-CAT and was very effective. I was very surprised by the model as the final F1 score of Loc-CAT was actually higher than the top competitor in the Kaggle Leader board. Now I did not expect to go back to the Proteins dataset as I was a bit intimidated by their 0.72 F1 score, however I found my motivation at the end of the paper where they said that they used an Adam optimizer and binary crossentropy as their loss function. It was at that lined that I realized the problems with my model and decided to go back.

When I went back to the two files that I was using for each of the models for the proteins dataset, I saw a lot of problems. For example, the CNN had no Dropout, so it was prone to over fitting. I was also using the wrong activation function for the last layer. For multi-label a sigmoid activation is used over a softmax function as the sigmoid function gives a confidence value for each of the classes separately whereas softmax's confidence values all add up to one. Another change that had to be done was to change the loss function and the optimizer of the model. In the previous iteration, I was using categorical crossentropy as the loss function and Stochastic Gradient Descent(SGD) for my optimizer. Categorical crossentropy is also called softmax loss and is generally used for multiclass classification whereas binary crossentropy is the prefered loss function for multilabel classification . For the optimizer, I was used to starting off with SGD for my optimizer and was not familiar with others, so after reading the paper, I decided to try out Adam.

After making all of my modifications, it was time to retrain the model. For training, I set the epochs to 10 and the `steps_per_epoch` to the length of the training data divided by the batch size, which came out to 873. I then let the model train for a total time of 24 hours. After training I was actually surprised to see what happened. Figure 1, shows the original attempt at the proteins dataset while Figure 2 shows the recent attempt. As Figure 1 shows, the first time that I trained the model, the training values were all over the place

4.8. **Rationale for Progress Journal #12**

I chose this progress journal because it marked a week where I had discovered many key findings. During this week, I was able to gather a lot of data to help support the fact that the CNNLSTM was better than the CNN. In the beginning, it would take me a week and a half to train both networks on one dataset, so I thought that adding three more datasets would have taken me the rest of the time that I had. However, I was a bit surprised, and I still am, that I was able to get both model trained and tested on all three of the datasets. This really boosted my confidence as with the amount of work that I was able to get done, I knew that I could carry this momentum back into the Proteins dataset. I was able to get usable data, that is being used throughout this entire portfolio. The only thing that I regret from this week was testing out the multi-output versions of each of the networks. It was not hard implementing this version, but the data that it produced was very cluttered. The last figure in the journal even shows this, as it is very hard to read when compared to the other graphs. Not to mention, the data did not tell me much as I knew that there was a bit of an uneven distribution within the classes, so some of the lines would not come out as smooth as the others. However, it was a bit hard to find these lines as they were hidden behind a bunch of other lines, and it was hard to separate the graphs. When I test both models, the multi-output model even did worse than the regular models. Looking back at this week, training both of the mult-output versions were my only regret. Even with this little regret, this week was one of the best that I had during the entire focus period. With all of the work that I had gotten done, I was able to gather enough data to help make my conclusion that the CNNLSTM was better than the CNN.

5. Summary Evaluation

5.1. What architecture is best for each model for a given task?

In all fairness, there is no definitive answer to this guiding question. Unfortunately, I was not able to step out of the VGGs and explore other architectures like I originally wanted to. However, throughout this process I have found that different architectures are helpful for different scenarios. For example, when I worked with the miml dataset, I found that the VGG11 worked better than the VGG16 and VGG19 since it had fewer parameters in it. Since the miml dataset did not contain too many images, an architecture with fewer parameters to train will be able to pick up on patterns easier. However, when the dataset gets larger like the Proteins dataset, it is better to use more complex architectures such as the VGG19, Loc-CAT, or even the Inception-Resnet (which I was not able to implement) as it would be able to see the patterns between classes better.

5.2. How do you implement these algorithms?

In order to implement these algorithms, you need two things: a programming language, and a machine learning library. There are a variety of programming languages that allow you to implement these algorithms. Go, Java, C, Octave, Matlab Javascript, and Python are very popular languages that support TensorFlow and other machine learning libraries. However, I chose to use Python as I was the most comfortable with the language. Python is also the most widely used language when it comes to implementing machine learning in the real world. In regards to machine learning libraries, the most popular happen to be TensorFlow and Keras, but many other libraries exist such as Theano and PyTorch.

5.3. What is the best way to implement these algorithms?

The two main libraries that I explored for this project Keras and TensorFlow. From all the work that I have done throughout the focus period, I feel that Keras was the best way to implement these algorithms. Keras is very developer friendly and is easy to use as a start. However, with more advanced algorithms, TensorFlow may be a better fit as it allows you to customize your algorithm better. However, TensorFlow is not very beginner friendly and it can be hard to get a grasp of. Therefore, I believe that Keras was the best way to implement

these algorithms as I did not have to do much customization and was able to debug these networks faster.

In terms of the programming languages, there really is no best language. Each language has its own quirks and advantages and disadvantages. When you really look at it, all it comes down to is your level of familiarity/comfort you have with the language. This reason is why I chose to implement these algorithms using Python as I had the most experience with the language and felt the most comfortable using it

5.4. How does one assess these networks?

The best metrics for assessing these networks are loss, accuracy scores, and statistical analysis. Loss allows you to see how well the model is doing over time. A high loss means that the model's weights are not optimized and as a result is the model can not make accurate classifications. One way to tell if a model is to see how the loss changes over time. If it decreases, the model is doing better over time and vice versa. Accuracy scores measure how many correct classifications the model has made. A higher accuracy score is desired as it means that the model's predictions are correct almost all of the time. Accuracy is computed by Keras automatically and is calculated using the following equation:

$$Accuracy = \frac{CorrectlyLabeledSamples}{TotalNumberofSamples} \tag{1}$$

Another popular metric to use is statistical analysis. In statistical analysis, the main metric that is looked at is the F1 Macro score. The F1 Macro Score is calculated by getting the averages of each of the classes and then taking the Macro average of these scores. In order to calculate F1, the following equation must be used:

$$F1 = 2 * \frac{Precision \times Recall}{Precision + Recall} \tag{2}$$

5.5. What is the difference between a multi-class and a multi-labeled dataset?

A multi-class dataset is a dataset that contains more than one possible classifications, but the contents of that dataset only belong to one class. For example, the MNIST dataset

is a multi-class dataset as each handwritten digit only represents one actual digit. On the other hand, a multi-labeled dataset is a dataset where one image could have more than one classification. For example, the Proteins dataset that I started with was a multi-label dataset as each picture had multiple classifications. When working with these datasets, it can be harder to work with a multi-labeled dataset than a multi-class dataset. This is because it can be harder to preprocess multi-labeled datasets than multi-class datasets. However, in the real world, a multi-labeled dataset is more common.

5.6. **How does each model do on a multi-class dataset?**

I tested both models on two different multi-class datasets: MNIST and FMINIST. The MNIST dataset is a collection of handwritten digits and is commonly used as a benchmark for many computer vision algorithms. For this dataset, the F1 scores and accuracy scores for both models for the testing data were very close. The CNN's F1 score was higher than the CNNLSTM as it scored a 0.99949 while the CNNLSTM's F1 score was 0.99932, but the difference between these two scores is minimal. On the other hand, the CNNLSTM was a bit more accurate than the CNN as the CNN scored a 0.9937 while the CNNLSTM scored a 0.9946. However, if we look at the trends during training, all of the metrics for the CNNLSTM were improving faster than the CNN and were all higher.

The FMNIST dataset also produced similar results. Since the dataset was introduced as an alternative to the MNIST dataset, I wanted to see how both models did on it. During testing, both models had the same F1 score, but the CNNLSTM had a higher accuracy score. However, the accuracy scores were very similar as the CNN scored a 0.9013 and the CNNLSTM scored a 0.9018. However, much like the MNIST dataset, the trends during training for the CNNLSTM were higher than the CNN and the training scores for the CNNLSTM increased faster.

5.7. **How does each model do on a multi-labeled dataset?**

The multi-labeled dataset was the main goal of this project as it encompassed the Proteins dataset. In order to test this guiding question, I trained both models on two different multi-labeled datasets: Multi-Instance and Multi-Label(miml) and the Human Protein Atlas. The

miml dataset was made by Nanjing University's Learning and Mining DatA center and contains multiple images that contain different combinations of deserts, mountains, sea, sunset, and trees. It was one of the only multi-label datasets that I could find. After training, the CNNLSTM actually did better than the CNN. The testing set for this dataset contained a 100 images. Out of the 100 images, all of the images had trees in them, 8 of the images had mountains and 1 had a picture of the sea. When they were ran on this set, the CNNLSTM got and f1 score of 0.435 and an accuracy score of 0.77 for the trees label while the CNN got a f1 score of 0.401 and an accuracy score of 0.67. The CNNLSTM also did better than the CNN for the mountains as it got a f1 score of 0.039 and an accuracy score of 0.375 the CNN got an f1 score of 0.012 and an accuracy score of 0.125.

Contrary to this, the Proteins dataset actually showed the CNN was better than the CNNLSTM. The dataset contains images that can have a combination of 28 different parts of a cell that are responsible for the production of proteins. These images come from the Human Protein Atlas and was meant as a challenging task. For the total accuracy scores, the CNN scored a 0.29 while the CNNLSTM scored a 0.19. Unfortunately, I could not get the final accuracy scores and f1 scores as Kaggle is still calculating them, but from what the accuracy scores from the validation set shows, the CNN actually did better than the CNNLSTM.

5.8. **How can these models be used in the real world?**

These models actually have a lot of real-world effects. Many of these models serve to automate large scale classification tasks, which allow people to move on and build upon what is known in the image. For example, a working model for the Proteins dataset could automate the entire classification task and speed up that process for researchers. These images allow researchers to understand protein function and could be used for other images to help understand diseases, so speeding up the classification task would be a huge benefit to these researchers.

5.9. **Closing Remarks**

When I started this project, I knew almost nothing about artificial neural networks and machine learning. I only knew the basic concepts from a Coursera course I took last year but did not know anything about how to really implement these models. That is why, in the beginning, much of my work was all over the place, as I was just trying to learn how I would go about implementing these models. However, it wasn't until the robotics side project failed that I started to focus in on what I set out to do.

Throughout the year, I was able to learn enough about neural networks and computer vision to actually implement my own algorithms. Along the way, I faced a lot of problems, but I was able to work through them and produce some work that I am very proud of. Through the focus program, I was able to learn a lot from how to use different python libraries, such as Keras and pandas, to which loss function to use for a given task. With all of the work that I have done, I was able to explore the world of machine learning and see just how fun it really is.