I went into model design with a general approach:

1. Start with a base model of one convolutional layer, max pool, and fully connected layer
2. Run the model to determine best optimiser, loss function, and learning rate for experimentation (setting a seed to standardise results).
3. Find a model construct that can achieve training accuracy to over 90%
4. Add further modifications to generalise the model performance to get testing accuracy as high as possible (optimally over 90%)

Based on the previous assignment that had an image classifier for character recognition, I was able to achieve >90% accuracy on a two-convolutional layer model, one fully connected layer, and a max pool. So, as a base, my model that determines cats will need to have at least two convolutional layers (as I believe there is more pattern recognition required to determine cats vs characters).

Initial model design that can be trained to achieve over 90% train accuracy:  
Convolutional Layer 1  
Max pool  
Convolutional Layer 2  
Max pool  
Fully connected layer  
Output layer

ReLU between the layers and Softmax in the output layer (Based on previous assignment and later observations of other CNN models).  
  
I knew cross entropy was probably the best loss function as the literature suggests that it is the best for multiclassification problems. I compared against nll\_loss which was much worse in performance.  
I tried various forms of SGA and RMSprop but ended up opting for Adam as an optimiser with a learning rate of 0.01 because it performed better.

Below is a list of changes I made in order based on experimentation:

* Added random vertical flip and horizontal flip transformations to improve generalisation
* Capped transformations at 25% as more started to inhibit training performance
* Model started to converge on only 47% test data, thought model would perform better being more sophisticated, added two convolutional layers (with max pools and drop out)
* Trained okay but still had poor test performance
* Added dropouts (0.25) after convolutional layers to improve generalisation
* Added weight decay (0.0001) to dampen large weights and improve generalisation
* Ended up crashing training performance
* After reading about weight decay interaction with learning rate, reduced learning rate to 0.001
* Training was too slow, added weight initialisation to train faster, opted for xavier uniform as it was apparently better suited for ReLu activations.
* Added a scheduler to improve performance and generalisation
* Test performance still low, added random rotation, colour gitter and random resized crop to transformations
* Upon reviewing data set, opted for 50% range on crop, based on images with dead space
* Upon further reflection, some of the cat images, the cat featured a very small portion of the image (far away with people in it) so opted to remove crop in case of random crops generating noise from cropping out cats.
* Increased hue volatility to max, reasoned that some cat breeds have high colour variation so I wanted my model to prioritise texture/shape recognition over colour relationships
* Train performance was still bad. Read about the construction of famous CNN models. Realised they had drop outs in fully connected layers, over convolutional layers. Reasoned that the drop outs in convolutional layer would inhibit some pattern recognition. Moved convolutional layers to fully connected layer (0.5). Added additional fully connected layer to improve computational speed.
* Realised famous CNN used fewer max pools, again reasoning it reduced pattern recognition. Removed from last two convolutional layers.
* With the removed max pools, training speed was much slower, removed a convolutional layer to reduce model sophistication (total 3).
* Increased out\_channel in first convolutional layer (and subsequent layers) to add sophistication to model without adding convolutional layer to improve accuracy.
* Model hit local minimum at 48%
* Model appears to require increased level of abstraction; however, 4 convolutional layers was too computationally heavy. Add extra two convolutional layers (total 5) with another pooling after second to reduce feature map while adding extra complexity. Also increase first kernel to 5. Lots of empty space to contend with in some images. Decreased the out\_channel in the first convolutional layer back to double (the output) to balance additional connections from additional convolutional layers.