

Convolutional Neural Network: Big Cat Image Classification

Holly Judge
Lizalise Luxande
Zahraa Hoosen

May 2023



Figure 1: Four Images from the Big Cats Dataset showing a Lion, a Leopard, a Jaguar and a Snow Leopard

1 Introduction

This project implements a Convolutional Neural Network (CNN) for a dataset of images of ten different species of big cats and it classifies these images into their corresponding classes to a certain degree of accuracy. This report starts with a description of the classification problem along with discussing its usefulness and ethics. It then explains the neural network model, model design decisions and the baseline used. The paper ends with a discussion on model validation and evaluation in terms of accuracy and loss.

2 Problem Description

Image classification is important as it allows a computer to understand visual information and categorise it just like a human [1]. A common task in machine learning and computer vision is multi-class image classification problems where the goal is to classify images into one of several predefined classes or categories [2]. By automatically categorising and labelling images into predefined classes, image classification facilitates efficient analysis of vast amounts of visual data [2]. Image classification have numerous real-world applications such as medical diagnosis, surveillance for security systems, face recognition and identifying animals - which this project will be focusing on for different species of big cats. Classifying images is difficult for computers but CNNs have been shown to have excellent performance at image recognition [3]. Thus, a CNN is ideal to use for classifying pictures of big cats.

A CNN is a type of deep learning neural network that is based off the structure of the brain [4]. CNNs excel in visual tasks due to their ability to learn hierarchical representations of visual features - lower layers capture simple features like edges and corners, while higher layers learn more complex features and concepts [2].

This project uses a CNN to classify pictures of big cats. The classifier will be able to predict the species of a big cat from taking in a photo of it and will return the name of the species it is (e.g. if it gets a picture of an ocelot, it should return ocelot). The inputs are images of ten different species of cats, and the outputs are the names of the ten different species of cats (such as puma, lion and cheetah).

The dataset was taken from Kaggle [5]. We preprocessed the data to ensure it had more variation in it so the model can generalise better later on. We then trained the neural network on this dataset (using the train, validate and test data) and experimented how different amounts of epochs led to different results. We then evaluated the trained model at the end of the project.

2.1 Usefulness

Building a CNN for classifying big cats can be useful in multiple ways. Firstly, it enables accurate species identification, contributing to species conservation and wildlife monitoring efforts [6]. By classifying different species of big cats from images or video footage, CNNs assist in population monitoring, habitat assessment, and conservation initiatives [7]. This information aids in understanding population dynamics, and the behaviour and habitat utilisation of big cats.

Secondly, CNNs for big cats can help support wildlife management and research activities [8]. By analysing camera trap images or satellite imagery, CNNs can automatically identify and track big cat species [6]. This data could contribute to ecological research, and help inform management strategies for protecting big cat populations and their habitats.

Furthermore, CNNs could also play a crucial role in anti-poaching and combating illegal wildlife trade [8]. They can analyse images or video data to identify illegal activities, detect poaching incidents, and recognize patterns related to illegal trade. CNNs can aid in anti-poaching efforts, support law enforcement actions, and help protect big cat populations from the threats of poaching and illegal trade.

Additionally, CNNs for big cats could have applications in ecotourism and education. Interactive tools or applications powered by CNNs can educate visitors about different big cat species, their conservation status, and their ecological importance. Such experiences enhance awareness, promote responsible tourism practices, and foster a deep understanding of the significance of protecting these endangered species.

The field of big cat classification remains significantly under-researched [8], emphasizing the substantial potential for impactful projects in this area. With limited existing research, the projects implemented have the opportunity to drive significant impact and advancements in big cat classification - thus, this is a useful problem to investigate.

By leveraging CNNs for big cat classification, we can enhance wildlife management, conservation planning, and contribute to the protection of them. CNNs can provide valuable tools for understanding, monitoring, and safeguarding big cat populations and their habitats.

2.2 Ethics

When building a CNN for classifying big cats, there are no significant ethical concerns regarding the dataset and the model. The dataset comprises images of big cats obtained from Google Images. There are two potential ethical issues worth considering for the dataset. Firstly, there is the question of photo ownership and informed consent for usage. As the photos were publicly available online, it is generally assumed that they can be used for non-commercial purposes without explicit consent.

The second ethical issue pertains to the respect for animals when capturing the photos. It is important to ensure that the images depict the animals in natural settings and that the animals were not caused distress or harm during the photography process. Given that the photos feature the animals in their natural habitats, it can be reasonably assumed that their welfare was respected.

There are also no ethical concerns regarding the development of an AI system for this task. It is unlikely bias will be introduced into this model that would make it unethical, and the results obtained from the system are not likely to be used in any malevolent or harmful manner.

Overall, while there are minor ethical considerations related to photo ownership in the dataset, building an AI system for classifying big cats does not pose any significant ethical issues.

2.3 Difficulty

This project encountered challenges that increased its difficulty. One of the key factors was the limited computational resources, resulting in prolonged training times. Since the project was executed on a laptop on its CPU and the lack of sufficient computational power led to extended training durations. The size of the dataset, consisting of approximately 2500 photos, further exacerbated these computational demands along with the complexity of the model (due to having multiple layers and pools) contributed to the need for extensive computational resources meaning training durations were long.

The large dataset with 2500 photos posed its own challenges in terms of data processing and model training. The difficulty of the project in terms of images was accounted for by including high-quality images in the dataset and applying various processing techniques. The images were preprocessed by rescaling, flipping, zooming (by 5%), and rotating (by 25%) at random, which enhanced the diversity and variability of the dataset. By increasing the range of variations in the training data, the model's performance and generalisation capabilities were improved.

Another difficulty encountered was figuring out the amount of epochs to use as some high amounts (such as epochs over 20) led to overfitting which reduced the accuracy of the model. Selecting the appropriate parameters for the layers, such as kernels and filters, also added an additional layer of complexity to the project.

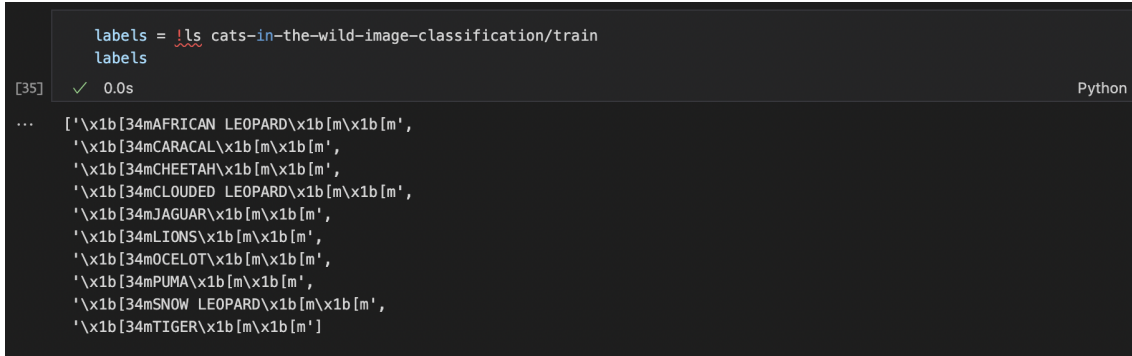
Overall, this project encompassed various elements that collectively heightened its difficulty. The combination of limited computational resources, a large dataset, complex model architecture, and the necessity of parameter tuning all contributed to the overall challenges faced.

3 Model Design

3.1 Project Set-up

The Convolutional Neural Network was designed and implemented using Keras, a high level neural network application programming interface, to build, train and evaluate the model on Jupyter Notebook.

The dataset obtained is publicly accessible from Kaggle. It includes images of 10 species of big cats, namely: African Leopard, Caracal, Cheetah, Clouded Leopard, Jaguar, Lions, Ocelot, Puma, Snow Leopard, and Tiger.



```

labels = !ls cats-in-the-wild-image-classification/train
labels
[35] ✓ 0.0s Python
... ['x1b[34mAFRICAN LEOPARD\x1b[m\x1b[m',
'x1b[34mCARACAL\x1b[m\x1b[m',
'x1b[34mCHEETAH\x1b[m\x1b[m',
'x1b[34mCLOUDED LEOPARD\x1b[m\x1b[m',
'x1b[34mJAGUAR\x1b[m\x1b[m',
'x1b[34mLIONS\x1b[m\x1b[m',
'x1b[34mOCELOT\x1b[m\x1b[m',
'x1b[34mPUMA\x1b[m\x1b[m',
'x1b[34mSNOW LEOPARD\x1b[m\x1b[m',
'x1b[34mTIGER\x1b[m\x1b[m']

```

Figure 2: Big Cat Classifications

The dataset is split into three sub-datasets for the training, validation and testing of the convolutional neural network:

1. Training Dataset: contains 2239 images (228-238 per the 10 image classes) with their corresponding labels
2. Validation Dataset: contains 50 images (5 per the 10 image classes) with their corresponding labels
3. Test Dataset: contains 50 images (5 per the 10 image classes) with their corresponding labels

3.2 Model design decisions

We decided to implement a convolutional neural network to handle image classification. This network is a feed-forward network where the first layer of the network is the input and the last layer is the output. The network includes more than one hidden layer making it a deep neural network.

3.3 Model Architecture

The architecture of the neural network includes input, hidden, and output layers. The input layer receives the initial data which is a processed image from the dataset. The model has multiple hidden layers of different types that perform specific tasks and operations to extract information from the input image:

- Convolutional layer(s): this layer has filters that extract valuable features from the image. This layer outputs a feature map that represents the specific feature in the input image.
- ReLU (Rectified linear Unit) layer(s): is an activation function that performs an operation that introduces the property of non-linearity to the learning model, given that the images are non-linear. This layer applies the activation function and outputs a rectified feature map.
- Pooling layer(s): is an operation that down samples the images. It is used to reduce the dimensions of the feature maps. It reduces the number of parameters to learn and reduces computation in the neural network. This layer generates a pooled feature map.

Lastly, the pooled feature map is flattened and the data is passed to a fully connected layer to get the output. The output layer is responsible for producing the network's outputs based on the learned knowledge and representations from the previous layers.

3.4 Baseline

The baseline for our model is a CNN with two convolutional layers each with 32 filters and with activation set to ReLu, two pooling layers of size 2 and a kernel size of 3. The number of epochs in our baseline is 11. The loss function used is categorical cross-entropy. There is no optimizer used. The accuracy of the baseline model is 44% and the loss is 1.47.

The baseline will be used as a reference to compare, and contextualise other trained models. The structure of the baseline is represented in the table below.

Model	Model Properties	Loss Function	Accuracy
Baseline	Number of epochs: 11 Filters: 32 Optimiser: none Activation Function: ReLU (Rectified Linear Unit) Loss Function: Categorical_cross-entropy 2D Convolution Layer(s): 2 Max Pooling 2D layer(s): 2	1.47	44%

Table 1: Baseline Model

4 Model Validation

4.1 Hyper-Parameter Tuning & Optimisation

We tested the neural network using hyper-parameter tuning to optimise the model and analyse its performance. The values of following hyper-parameters affect the performance and accuracy of the neural network when tuning the model:

- 2D Convolutional layers: the number of layers in the neural network has an impact on the learning capabilities, performance or the neural network. We will trial with 2 and 4 convolutional layers.
- Layer filters: refers to the filters that are applied to a specific layer in the neural network. We will trial models with 64 and 32 filters where each filter will detect a specific pattern or feature in the input data. After the operation, the layer will produce either 64 or 32 feature maps, respectively, depending on the number of filters.
- Kernel size: refers to the dimensions of the filter that moves over the input data during the convolution operation. In the model, it is a square matrix with a width and height. The choice of the kernel size depends on the characteristics of the input data. Generally, smaller kernel sizes are used to capture local features and larger kernel sizes are used to capture global features. For this specific classification, we will trial three different kernel sizes (2, 3 and 7) to find the optimal value.
- Epochs: an epoch refers to one cycle through the dataset. It is the number of times the entire dataset is passed forward and backward through the neural network. Various numbers of epochs (11, 15, 20, 25) will be trialled to find an optimal number.
- Optimiser: it is a function that adapts the neural network's attributes. It is an internal parameter of a model that aims to reduce the error and minimise the loss function. We will conduct tests/trials with and without the optimiser to determine how it affects the performance of the model. We make use of the Adaptive Moment Estimation (ADAM) optimiser.

The results of the hyper-parameter tuning are represented in the table below:

Trial	Number of Convolutional Layers	Number of filters	Kernel Sizes	Number of Epochs	Optimiser	Loss	Accuracy
1	2	32	2, 2	11	None	1.468	44%
2	2	64	2, 2	11	None	0.961	56%
3	2	64	3, 3	11	None	1.158	57%

4	2	32	3, 3	11	None	1.01	60%
5	4	32	3, 3, 3, 3	11	None	0.712	70%
6	4	32	3, 3, 3, 3	15	None	0.996	62%
7	2	32	3, 3	15	None	0.898	64%
8	4	32	3, 3, 3, 3	11	ADAM	0.860	74%
9	4	32	3, 3, 3, 3	20	ADAM	0.654	78%
10	4	32	3, 3, 3, 3	25	ADAM	0.8	68%
11	2	32	7, 7	11	ADAM	2.07	26%

Table 2: Hyper-parameter tuning

Trial 9 with 4 convolutional layers, 32 filters, kernel size set to 3 and number of epochs set to 20 led to the best results in terms of loss and accuracy (0.654 and 78% respectively) which will be discussed in evaluation below.

5 Evaluation

The dataset used was divided into three parts: a training set of 2239 images, a validation set of 50 images and a test dataset of 50 images. The purpose of the validation set was to judge how well the model could generalise at each epoch, as a high accuracy value associated with a lagging validation accuracy value indicates the model is over-fitting.

5.1 Evaluation metrics

A combination of two evaluation metrics are used to quantify the performance and measure the quality of the neural network machine learning model.

Accuracy: measures the proportion of correct predictions made by the model out of the total number of predictions. It is represented as a percentage and a higher accuracy percentage correlates to a more accurate model that makes a higher number of correct predictions (out of the total number of predictions).

Loss: measures the discrepancy between the predicted output and the true target values of the model. More specifically, the model uses the categorical cross-entropy loss function. This loss function is used in multi-class classifications such as this classification model. It measures the dissimilarities between the predicted class probabilities and the true class labels. A lower cross-entropy loss indicates a better model fit. The goal is to minimise the cross-entropy loss.

5.2 Model evaluation

Our baseline model is a CNN with 2 layers and 32 filters per layer, a kernel of size 2 x 2, 11 epochs and no optimizer. The first parameter we tuned was the number of filters. Increasing the number of filters from 32 to 64 increased the model's accuracy and decreased the loss value. The kernel size was modified to 3 which again increased the performance of the model. Trials 1 to 4 demonstrated that the combination of 32 filters per layer and a kernel size of 3 x 3 was performing best. Therefore, these two parameters remained fixed in subsequent trials.

In trial 5 we increased the number of layers in our CNN from 2 to 4. The effect was improving the accuracy score by 10% and decreasing the loss value by 0.298. In trial 6 the number of epochs increased from 11 to 15. The result was the model over-fitting, demonstrated by the decrease in accuracy and loss. The over-fitting was addressed by using the optimizer ADAM. ADAM allowed for the epoch value to increase without causing over-fitting. The result was the model reaching its peak performance at 20 epochs, using ADAM as an optimizer, with 4 convolutional layers, 32 filters per layer and a kernel size of 3 x 3. At epoch 25 the model began to overfit again.

Trial 12 was done to understand the effect of increasing the kernel size on accuracy and loss in the model.

5.3 Analysis of model performance

The model improved from an accuracy of 44% and a loss value of 1.468 to an accuracy of 78% and a loss value of 0.654. The hyper-parameter adjustments made thus improved the accuracy by 77% and decreased the loss value by over 55%. Increasing

Training set after final epoch	Validation set after final epoch	Test set after final epoch
65%	52%	44%

Table 3: Baseline model accuracy

Training set after final epoch	Validation set after final epoch	Test set after final epoch
1.039	1.629	1.468

values

Table 4: Baseline model loss

Training set after final epoch	Validation set after final epoch	Test set after final epoch
83%	60%	78%

Table 5: Optimal model accuracy

Training set after final epoch	Validation set after final epoch	Test set after final epoch
0.494	1.109	0.654

Table 6: Optimal model loss values

```
bigcat_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32,
                           kernel_size=3,
                           activation="relu",
                           input_shape=(224, 224, 3)),
    tf.keras.layers.Conv2D(32, 3, activation="relu"),
    tf.keras.layers.MaxPool2D(pool_size=2,
                              padding="valid"),
    tf.keras.layers.Conv2D(32, 3, activation="relu"),
    tf.keras.layers.Conv2D(32, 3, activation="relu"),
    tf.keras.layers.MaxPool2D(2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(num_of_classes, activation="softmax")
])

bigcat_model.compile(loss="categorical_crossentropy",
                    optimizer=tf.keras.optimizers.legacy.Adam(),
                    metrics=["accuracy"])

bigcat_history = bigcat_model.fit(train_data,
                                  epochs=11,
                                  steps_per_epoch=len(train_data),
                                  validation_data=valid_data)
```

Figure 3: Optimal Model

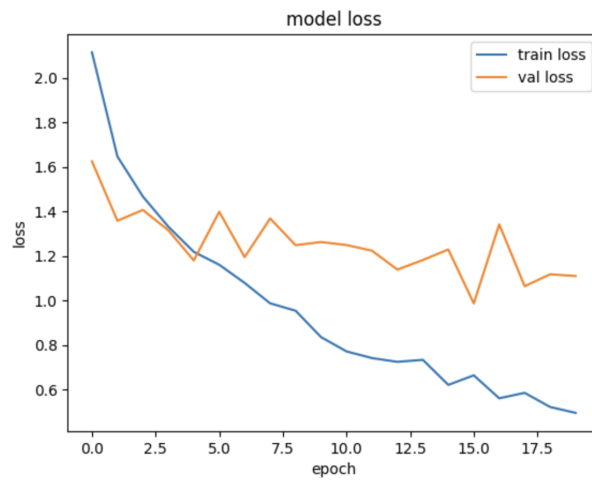


Figure 4: Model Loss of Optimal Model

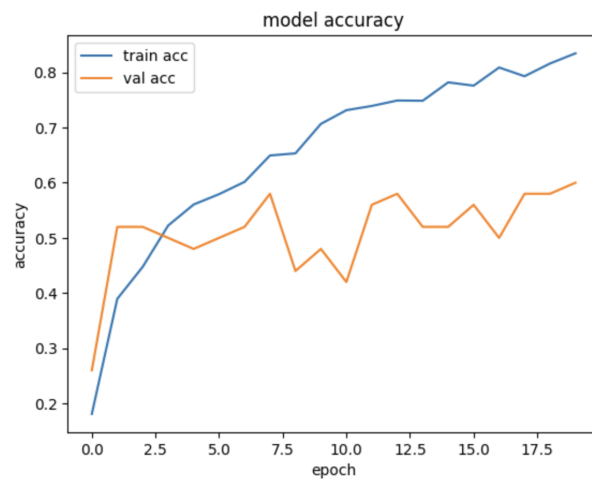


Figure 5: Model Accuracy of Optimal Model

6 Conclusion

In this report, we created a CNN that classifies big cat images into their respective species classes. Initially we created a model with 2 layers, 32 filters per layer, a kernel size of 2 x 2 and 11 epochs. However, our baseline model with basic architecture performed poorly with an accuracy of 44% and loss of 1.468. Using a kernel size of 3 x 3 and using 4 layers instead of 2 layers, our model was able to reach 70% accuracy. After this point, any increase in epoch size decreased the accuracy and increased the loss. To combat this we introduced an optimizer, ADAM. Using ADAM we were able to reach our optimal model. Our optimal model (trial 9) is a 4 layer CNN, with 32 filters per layer, 3 x 3 kernels, and 20 epochs. Our optimal model has an accuracy of 78% and a loss value of 0.654. Overall our optimal model performed decently with an looking at its accuracy.

Future work could look at increasing the number of species in the dataset, at individual animal identification within specific species or at fine-tuning the existing model. It would be also interesting to see different animal species trained using this model, if the results would be similar in terms of accuracy or how would they differ. Exploring the application of this model to train on different animal species would be interesting, as it would provide insights into the comparability of accuracy of this model across different animal species.

References

- ¹N. Tariq, K. Saleem, M. Mushtaq, and M. A. Nawaz, “Snow leopard recognition using deep convolution neural network”, in Proceedings of the 2nd international conference on information system and data mining (2018), pp. 29–33.
- ²K. O’Shea and R. Nash, “An introduction to convolutional neural networks”, arXiv preprint arXiv:1511.08458 (2015).
- ³I. Jarraya, F. BenSaid, W. Ouarda, U. Pal, and A. M. Alimi, “A new convolutional neural network based on a sparse convolutional layer for animal face detection”, Multimedia Tools and Applications **82**, 91–124 (2023).
- ⁴H. Huang, “Convolutional neural network-based image recognition for animals”, in Proceedings of the 3rd international symposium on artificial intelligence for medicine sciences (2022), pp. 123–127.
- ⁵Gerry, *10 big cats of the wild - image classification*, Feb. 2023.
- ⁶A. M. Roy, J. Bhaduri, T. Kumar, and K. Raj, “A computer vision-based object localization model for endangered wildlife detection”, Ecological Economics, Forthcoming (2022).
- ⁷M. Timm, S. Maji, and T. Fuller, “Large-scale ecological analyses of animals in the wild using computer vision”, in Proceedings of the ieee conference on computer vision and pattern recognition workshops (2018), pp. 1896–1898.
- ⁸L. Feng, Y. Zhao, Y. Sun, W. Zhao, and J. Tang, “Action recognition using a spatial-temporal network for wild felines”, Animals **11**, 485 (2021).

7 Appendix

Please see documents attached for the trial test results, and instructions reproduce results

7.1 Project README

The project ReadMe entails instruction of how to produce results for the baseline and optimal model, and of other trials/tests with the desired hyper-parameters

AI ASSIGNMENT 2: PROJECT README

About the Project The project develops a convolutional neural network AI system that classifies ten categories of big cats, and evaluates the performance of the model.

Built with Keras open-source software: library that provides a Python interface for artificial neural networks
Jupyter Notebook: computing platform

Instructions to produce the Baseline model

- Use existing code base to train the model:

- Use two 2-D convolutional layers
- Use two max-pooling layers
- Set Filters to 32
- Set kernel_size to 2
- Set all activations to 'relu'
- Set input_shape to (224, 224, 3)
- Set pool_size to 2
- Set padding to valid
- Set Dense layer activation to 'softmax'
- Remove optimizer from compile code
- Set the number of epochs to 11
- Run code

Instructions to produce the Optimal model

- Use existing code base to train the model:
- Use four 2-D convolutional layers
- Set Filters to 32
- Set kernel_size to 3
- Set all activations to 'relu'
- Set input_shape to (224, 224, 3)
- Set pool_size to 2
- Set padding to valid
- Set Dense layer activation to 'softmax'
- Set optimizer to 't.f.keras.optimizers.legacy.Adam'
- Set the number of epochs to 20
- Run code

Instructions to produce a trial/test

Alter the desired hyper-parameters from the code base and run the program

7.2 Trial test results

```
Epoch 1/20
2023-05-24 17:47:45.200750: I tensorflow/core/common_runtime/executor.cc:1210] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message):
INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
[[{{node Placeholder/_0}}]]
74/74 [=====] - ETA: 0s - loss: 2.1143 - accuracy: 0.1808
2023-05-24 17:48:43.743158: I tensorflow/core/common_runtime/executor.cc:1210] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message):
INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
[[{{node Placeholder/_0}}]]
74/74 [=====] - 59s 790ms/step - loss: 2.1143 - accuracy: 0.1808 - val_loss: 1.6246 - val_accuracy: 0.2600
Epoch 2/20
74/74 [=====] - 63s 844ms/step - loss: 1.6463 - accuracy: 0.3895 - val_loss: 1.3576 - val_accuracy: 0.5200
Epoch 3/20
74/74 [=====] - 63s 854ms/step - loss: 1.4665 - accuracy: 0.4476 - val_loss: 1.4062 - val_accuracy: 0.5200
Epoch 4/20
74/74 [=====] - 65s 873ms/step - loss: 1.3309 - accuracy: 0.5224 - val_loss: 1.3147 - val_accuracy: 0.5000
Epoch 5/20
74/74 [=====] - 67s 902ms/step - loss: 1.2179 - accuracy: 0.5605 - val_loss: 1.1793 - val_accuracy: 0.4800
Epoch 6/20
74/74 [=====] - 65s 877ms/step - loss: 1.1598 - accuracy: 0.5793 - val_loss: 1.3980 - val_accuracy: 0.5000
Epoch 7/20
74/74 [=====] - 66s 895ms/step - loss: 1.0780 - accuracy: 0.6015 - val_loss: 1.1940 - val_accuracy: 0.5200
Epoch 8/20
74/74 [=====] - 66s 890ms/step - loss: 0.9865 - accuracy: 0.6494 - val_loss: 1.3679 - val_accuracy: 0.5800
Epoch 9/20
74/74 [=====] - 68s 922ms/step - loss: 0.9532 - accuracy: 0.6533 - val_loss: 1.2477 - val_accuracy: 0.4400
Epoch 10/20
74/74 [=====] - 69s 939ms/step - loss: 0.8347 - accuracy: 0.7067 - val_loss: 1.2620 - val_accuracy: 0.4800
Epoch 11/20
74/74 [=====] - 67s 903ms/step - loss: 0.7704 - accuracy: 0.7315 - val_loss: 1.2484 - val_accuracy: 0.4200
Epoch 12/20
74/74 [=====] - 69s 931ms/step - loss: 0.7406 - accuracy: 0.7392 - val_loss: 1.2231 - val_accuracy: 0.5600
Epoch 13/20
74/74 [=====] - 67s 910ms/step - loss: 0.7236 - accuracy: 0.7490 - val_loss: 1.1381 - val_accuracy: 0.5800
Epoch 14/20
74/74 [=====] - 67s 899ms/step - loss: 0.7327 - accuracy: 0.7486 - val_loss: 1.1812 - val_accuracy: 0.5200
Epoch 15/20
74/74 [=====] - 68s 913ms/step - loss: 0.6201 - accuracy: 0.7820 - val_loss: 1.2286 - val_accuracy: 0.5200
Epoch 16/20
74/74 [=====] - 71s 956ms/step - loss: 0.6630 - accuracy: 0.7760 - val_loss: 0.9861 - val_accuracy: 0.5600
Epoch 17/20
74/74 [=====] - 68s 930ms/step - loss: 0.5598 - accuracy: 0.8089 - val_loss: 1.3414 - val_accuracy: 0.5000
Epoch 18/20
74/74 [=====] - 68s 921ms/step - loss: 0.5843 - accuracy: 0.7931 - val_loss: 1.0635 - val_accuracy: 0.5800
Epoch 19/20
74/74 [=====] - 68s 917ms/step - loss: 0.5207 - accuracy: 0.8162 - val_loss: 1.1170 - val_accuracy: 0.5800
Epoch 20/20
74/74 [=====] - 72s 972ms/step - loss: 0.4943 - accuracy: 0.8345 - val_loss: 1.1091 - val_accuracy: 0.6000

2023-05-24 18:10:01.922545: I tensorflow/core/common_runtime/executor.cc:1210] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message):
INVALID_ARGUMENT: You must feed a value for placeholder tensor 'Placeholder/_0' with dtype int32
[[{{node Placeholder/_0}}]]
2/2 [=====] - 1s 134ms/step - loss: 0.6542 - accuracy: 0.7800
[0.6541502475738525, 0.7799999713897705]
```

Figure 6: Optimal model (Trial 9) results