ZZEN9444 Neural Networks - Assignment 3 Report

By Nidhi Wadhwa

Aim

The aim of this project was to find and train the most accurate neural network model on the provided 8000 images of cats and categorise each image into its relevant group that corresponds to the type of cat.

Method

A series of steps and experiments were followed and trialled to train the most accurate neural network on the 8000 images.

Step 1. Initial Model Architecture Selection

Decision 1: Make use of pre-trained models

The approach selected was to make use of pre-trained models and the pre-trained weights and fine-tune these models on the provided 8000 image dataset.

The decision to use pre-trained models and pre-trained weights was based on a number of factors:

1. Saving time and compute power: Pre-trained models have already learned general features (for image classification these include features like edges, textures, shapes, etc.).
2. Better model performance: Given pre-trained models have been trained on large datasets, they often perform better when fine-tuned on a small dataset rather than training a model from scratch with a small dataset.
3. Leverage state-of-the-art model architectures: Using pre-trained models provides the benefit of being able to use state-of-the-art model architectures that have been extensively tested and optimized.
4. Leverage pre-trained weights

Decision 2: Which pre-trained models to use

There's an extensive list of pre-trained pytorch models for image classification - https://docs.pytorch.org/vision/stable/models.html. These are all popular pre-trained Convolutional Neural Network models.

CNN's are good for image classification as they:

- Can detect local patterns like edges, textures and shapes
- Have hierarchical feature learning, where early layers can detect low-level features like edges and colours, and deeper layers learn high-level features like various objects (both object parts and full objects)

By analysing the reported accuracies of the pre-trained models on the ImageNet-1k dataset (models with highest accuracies) as well as the size of the models (final trained model needs to be less than 50MB), the following models were selected for experimentation for this project:

- MobileNet_V3_Large_Weights.IMAGENET1K_V2
- EfficientNet_B0_Weights.IMAGENET1K_V1
- EfficientNet_B3_Weights.IMAGENET1K_V1
- MNASNet1_3_Weights.IMAGENET1K_V1
- RegNet_Y_800MF_Weights.IMAGENET1K_V2
- ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1

The following table provides a brief overview of each model and their respective architectures. Note: the details were purposely kept at a high-level, as the model architectures are quite complex and large.

Model	**Architecture Overview**	**Key Architectural Components**
MobileNet_V3_Large_Weights.IMAGENET1K_V2	Lightweight CNN, optimised for mobile and edge devices	<u>Stem Layer</u> - Standard convolution: 3×3, stride 2, output channels = 16. <u>Inverted Residual Blocks</u> - Expansion layer (1×1 conv) - Depthwise convolution (3×3 or 5×5) - Projection layer (1×1 conv) - Optional SE block - Activation: ReLU or Hard-Swish - Strides vary (1 or 2) for downsampling. <u>Classifier</u> - 1×1 Conv → Global Avg Pool → Dropout → FC
EfficientNet_B0_Weights.IMAGENET1K_V1	Efficient CNN, optimised for efficiency and accuracy	<u>Stem Layer</u> - Conv3×3, stride 2, output channels = 32. <u>MBConv Blocks</u> - Inverted residual structure - Depthwise convolution - SE block for channel attention - Expansion and projection layers - Activation: Swish - Strides vary for downsampling. <u>Classifier</u> - 1×1 Conv → Global Avg Pool → Dropout → FC

EfficientNet_B3_Weights.IMAGENET1K_V1	Scaled-up EfficientNet variant with higher depth, width, and resolution	Same core design as B0, but scaled: - Larger MBConv blocks with SE - More layers and channels - Input resolution: 300×300 - Activation: Swish Classifier - 1×1 Conv → Global Avg Pool → Dropout → FC
MNASNet1_3_Weights.IMAGENET1K_V1	CNN optimised for mobile devices	Stem Layer - Conv3×3, stride 2. Inverted Residual Blocks - Expansion layer (1×1 conv) - Depthwise convolution (3×3) - Projection layer (1×1 conv) - Activation: ReLU - No SE by default - Strides vary for downsampling. Classifier - 1×1 Conv → Global Avg Pool → FC

RegNet_Y_800MF_Weights.IMAGENET1K_V2	CNN, part of RegNet family: optimised for a trade-off between accuracy and efficiency	Stem Layer
		- Conv3×3, stride 2.
		Stages
		- Bottleneck blocks with:
		  - 1×1 Conv (reduce)
		  - 3×3 Grouped Conv
		  - 1×1 Conv (expand)
		- SE block for channel attention
		- Activation: ReLU
		- Width increases linearly across stages.
		Classifier
		- Global Avg Pool → FC

ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1	Ultra-efficient CNN optimised for efficiency and low latency, very fast on mobile	Stem Layer
		- Conv3×3, stride 2.
		Shuffle Blocks
		- Two-branch structure:
		  - One identity branch
		  - One branch: 1×1 Conv → Depthwise Conv → 1×1 Conv
		- Channel split and channel shuffle for cross-group info
		- Activation: ReLU
		- Strides vary for downsampling.
		Classifier
		- 1×1 Conv → Global Avg Pool → FC

Table Legend:

- Stem Layer: First processing layer in network (after initial input layer)
- SE: Squeeze-and-Excitation - type of channel attention mechanism
- FC: Fully Connected (layer)
- MBConv: Mobile Inverted Bottleneck convolution - type of building block in CNN's, combination of specific layers
- Inverted Residual block - type of building block in CNN's, combination of specific layers

Step 2. Initial Model Design and Parameter Selection

To trial how the various selected pre-trained models perform on the 8000 image dataset, some initial model design and parameter decisions were made.

Decision 1: Initial Optimizer Function

The use of a variant Adam optimizer - AdamW - was initially selected as the optimizer function. Adam is a popular, widely used optimizer function as it utilises adaptive learning rates, often has fast convergence without much need for tuning or heavy hyperparameter tuning.

Below is a brief comparison of some Adam variants:

Adam Variant	**Advantages**	**Disadvantages**	**When to Use**
Adam (original)	- Faster convergence from use of adaptive learning rate and adaptive momentum	- Use of weight decay (L2 regularization) can lead to suboptimal generalisation	- Good for quick experimentation - Good when weight decay is not critical
AdamW	- Improved generalisation over original Adam by decoupling weight decay from gradient-based updates - Used in many state-of-the-art models	- Requires more hyperparameters to tune due to additional complexity of weight decay as a separate parameter	- Require effective weight decay regularization - Want better performance and generalisation
AMSGrad	- Ensures adaptive learning rate doesn't increase, almost guaranteeing convergence in particular setups - Helps stabilise training	- Usually doesn't outperform AdamW when applied	- Good to use if having issues with Adam's convergence or stability - often in sensitive or complex training situations

AdamW was selected between these Adam variants due to requiring a model with good generalisation and its popularity in state-of-the-art models.

Stochastic Gradient Descent is another popular optimizer function. This was also explored throughout this project - see in Step 7.

Decision 2: Learning Rate

The learning rate was initially set to 0.0001. This learning rate is generally a good initial learning rate for transfer learning, allowing for effective adaption of pre-trained features, generally allows for stable training and works well with the selected optimizer AdamW.

The use of a learning rate scheduler was explored in this project - see in Step 4.

Decision 3: Weight Decay

An initial value of 0.01 was used for weight decay. While the use of weight decay is not required, it was selected for use to:

- Help prevent overfitting by encouraging smaller weights in the model
- Improve generalisation of the model by keeping weights small
- stabilise training by helping avoid weights becoming too large

Typical values of weight decay range from 0.01 to 0.0001. Initially, the value of 0.01 was selected to be more aggressive to prevent overfitting.

Other values of weight decay were also explored in this project - see in Step 6.

Decision 4: Loss Function

Cross-entropy loss is a common loss function used for image classification. Cross-entropy measures the difference between the predicted probability distribution and the true distribution, encouraging a model to assign a high probability to the correct class and low probability to others. Cross-entropy loss also works well with common optimization functions like Adam and SGD. Hence, it was used as the loss function in this project.

Decision 5: Batch Size

Three batch sizes were initially experimented with - 64, 128 and 200. This was for two main reasons:

1. With larger batch sizes, some pre-trained models required a high GPU RAM size. As this was a constraint within the project (limits on GPU RAM size), smaller batch sizes allowed to trial of more pre-trained models.
2. More model generalisation and more stable training - Smaller batch sizes can allow for better model generalisation and can be better for fine-tuning. Larger batch sizes can allow for more stable gradients and hence training and faster training per epoch, but can sometimes lead to overfitting. Hence, both approaches were experimented with.

Decision 6: Number of Epochs

Three different epoch sizes were initially experimented with - 10, 20 and 30. This was to account for batch size - with use of smaller batch sizes, the number of epochs were sometimes required to be larger to account for slower training. However, too many epochs also resulted in overfitting, so sometimes smaller epoch sizes proved to perform better.

Decision 7: Training & Validation Set Split

A validation set was used to help with overfitting. By monitoring the validation loss and accuracy, it becomes easier to observe if the model is memorising or generalising the training data. Initially a training / validation set split of 0.8 was used. This value was selected as it provides a good balance between having enough data for training, so a model can generalise well, and having enough data in the validation set to reliably estimate the model's performance when tuning hyperparameters and to check for overfitting.

A training / validation set split of 0.9 was also later experimented on - see Step 9.

Decision 8: Data Transformation

Two image transformations were initially used - re-scaling images and normalization.

1. Re-scaling Images

The original dataset images were required to be re-scaled to sizes that the pre-trained models would expect. This varied per pre-trained model, but common re-scaling sizes were either (224, 224) or (300, 300)).

2. Normalisation

Normalisation is a common technique of data transformation that helps ensure models train efficiently and properly converge. Normalisation involves normalising the data pixel values by subtracting the value 255, converting the pixel values to values between 0 and 1, subtracting the dataset's mean and dividing by its standard deviation per (R, G, B) channel.

Standardising / normalising data inputs to neural networks often helps keep values within a consistent range, helping gradients not vanish or explode, and hence stabilising training. Normalising can also help the model converge faster, as well as improving generalisation of the model.

Furthermore, the pre-trained models used in this project are trained on normalised data (using the ImageNet mean and standard deviation values). Not using normalisation will result in performance of the model dropping.

Three experiments were conducted to select if normalisation should be used and which values to use for normalisation. The following are the experiments conducted:

<u>Experiment 1:</u> No data normalisation

Experiment 2: ImageNet normalisation

For ImageNet-pre-trained modes, the commonly used mean and standard deviation values per colour channels (Red, Green, Blue) are:

Mean: [0.485, 0.456, 0.406]

Standard Deviation: [0.229, 0.224, 0.225]

These means and standard deviation values have been computed from the ImageNet training dataset.

Experiment 3: Actual dataset value normalisation

The actual mean and standard deviation for the 8000 image dataset was independently calculated. Using these values was experimented with in hope that performance would improve in case the dataset has a different pixel distribution in comparison to ImageNet.

The calculated mean and standard deviation for the dataset were as follows:

Mean: [0.4806, 0.4338, 0.3918]

Standard Deviation: [0.2422, 0.2343, 0.2270]

It can be observed that these values are fairly close to ImageNet's mean and standard deviation.

Experimentation Results

To test these experiments, one pre-trained model - MobileNet_V3_Large_Weights.IMAGENET1K_V2 - was selected. The batch size was selected at 200 and number of epochs at 10. The other parameters were as defined in the previous sections.

The results from the three experiments are summarised in the following table:

Model	**Normalisation**	**Results**
MobileNet_V3_Large_Weights .IMAGENET1K_V2	Without any normalisation data transformation	ep 10, loss: 0.21, 6400 train 99.91%, 1600 test 87.81%
MobileNet_V3_Large_Weights .IMAGENET1K_V2	Imagenet normalisation data transformation	ep 10, loss: 0.21, 6400 train 99.89%, 1600 test 90.88%
MobileNet_V3_Large_Weights .IMAGENET1K_V2	Results with dataset normalisation	ep 10, loss: 0.22, 6400 train 99.89%, 1600 test 91.81%

Based on these results, using a normalisation data transformation seems to improve model performance. From these findings, using the actual dataset's mean and standard deviation values was selected to be used for normalisation for further experimentation.

Other Data Transformation Alternatives

Data augmentation is another data transformation that could be applied. Data augmentation includes applying transformations to data like changing the geometry (rotation, flipping, etc.), altering colour, noise and more. These transformations can help to increase the size and diversity of a dataset, helping prevent a model from overfitting and improves generalisation. However, due to this project's time constraints, this was not explored in this project.

Step 3. Initial Selected Models Experimentation

Based on the initial design and parameter choices defined in Step 2, a number of experiments were conducted on the selected pre-trained models.

When training and fine-tuning these models, it was decided not to freeze any layers in the pre-trained models and to train all models for best fine-tuning results.

The results from these experiments are summarised in the following table, including the pre-trained model, the batch size, number of epochs and the final results produced.

Experimentation Results

Model	**Batch Size**	**Epochs**	**Results**
MobileNet_V3_Large_Weights.IMAGENET1K_V2	200	10	ep 10, loss: 0.22, 6400 train 99.89%, 1600 test 91.81%
MobileNet_V3_Large_Weights.IMAGENET1K_V2	64	10	ep 10, loss: 0.87, 6400 train 99.81%, 1600 test 90.00%
MobileNet_V3_Large_Weights.IMAGENET1K_V2	64	30	ep 30, loss: 1.12, 6400 train 99.62%, 1600 test 88.25%
MobileNet_V3_Large_Weights.IMAGENET1K_V2	128	30	ep 30, loss: 0.37, 6400 train 99.70%, 1600 test 88.88%
EfficientNet_B0_Weights.IMAGENET1K_V1	64	30	ep 30, loss: 0.93, 6400 train 99.77%, 1600 test 90.94%
EfficientNet_B0_Weights.IMAGENET1K_V1	64	30	ep 30, loss: 1.03, 6400 train 99.66%, 1600 test 92.00%

MNASNet1_3_Weights.IMAGENET1K_V1	200	10	ep 10, loss: 0.16, 6400 train 99.95%, 1600 test 80.06%
MNASNet1_3_Weights.IMAGENET1K_V1	64	20	ep 20, loss: 0.74, 6400 train 99.83%, 1600 test 80.50%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	200	10	ep 10, loss: 0.15, 6400 train 99.94%, 1600 test 89.44%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	ep 20, loss: 0.25, 6400 train 99.94%, 1600 test 91.50%
ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1	200	10	ep 10, loss: 0.20, 6400 train 99.94%, 1600 test 89.88%
ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1	64	10	ep 10, loss: 0.58, 6400 train 99.91%, 1600 test 91.56%
EfficientNet_B3_Weights.IMAGENET1K_V1	64	10	ep 10, loss: 3.53, 6400 train 99.50%, 1600 test 92.06%

Based on these results, the top performing models with their respective parameter values were:

Model	**Batch Size**	**Epochs**
MobileNet_V3_Large_Weights.IMAGENET1K_V2	200	10
EfficientNet_B0_Weights.IMAGENET1K_V1	64	30
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20
ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1	64	10
EfficientNet_B3_Weights.IMAGENET1K_V1	64	10

These models were selected to be used for further fine-tuning and experimentation. However the model EfficientNet_B3_Weights.IMAGENET1K_V1 was decided not to be used for further experimentation as training of this model took a long amount of time due to its size and experimentation time was limited in this project.

Step 4. Learning Rate Schedular Experimentation

Selecting a good learning rate value is important for a model to properly converge and lead to better model generalisation. The choice of a poor learning rate can lead to suboptimal model performance, where either overfitting or underfitting of a model is common.

The use of a learning rate scheduler was experimented with. Learning rate schedulers automatically adjust the learning rate during training, allowing for higher learning rates in the early stages of training so as to not get stuck in local minimas and allowing for lower learning rates in later stages of training to allow for fine-tuning and convergence to optimal performance, without allowing optimizer functions from missing any minimums.

There are a number of different learning rate schedulers. A quick summary of various schedulers and when to use them is as follows:

- CosineAnnealingLR: Good default choice, works well for transfer learning
- StepLR: When specific learning rate reductions should be made at known epochs
- ReduceLROnPlateau: When there's validation data and want adaptive scheduling
- MultiStepLR: When longer training is required with a predetermined schedule
- ExponentialLR: When a more gentle and consistent decay is desired

Out of these options, the CosineAnnealingLR learning rate scheduler was selected for the following reasons:

- Works well for transfer learning when fine-tuning pre-trained models
- Has smoother convergence and avoids training instability from sudden learning rate drops
- Often works well without hyperparameter tuning
- Widely used in state-of-the-art models

This learning rate scheduler was used with the following parameters:

scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs, eta_min=1e-6)

The eta_min value is the minimum learning rate that the scheduler will go down to. For use with an Adam optimizer function, having a small non-zero value for this parameter is common, at a value of the initial learning rate * 1e-2 or 1*e-3. Hence, the value 1e-6 was selected.

Experiments of the use of the learning rate scheduler were trialled on the shortlisted models per Step 3.

For experimentation purposes, the optimizer function in this section was kept as AdamW as per Step 2.

The number of epochs were as per the models and their respective epoch numbers were selected per the shortlisted models in Step 3.

The initial learning rate was kept as 0.0001 as per Step 2. This initial learning rate was selected as a good choice of learning rate to allow for effective adaptation of pre-trained features, to allow for stable training, a learning rate value that won't destroy the pre-trained ImageNet features that the pre-trained models already implement and works well with the optimizer function AdamW.

Experiments using different learning rate values were also explored in this project - see Step 8.

Experimentation Results

Model	**Batch Size**	**Epochs**	**Results**
EfficientNet_B0_Weights.IMAGENET1K_V1	64	30	ep 30, loss: 0.56, 6400 train 99.84%, 1600 test 91.25%
MobileNet_V3_Large_Weights.IMAGENET1K_V2	64	20	ep 20, loss: 0.17, 6400 train 99.98%, 1600 test 91.44%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	ep 20, loss: 0.16, 6400 train 100.00%, 1600 test 93.88%
ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1	64	30	ep 30, loss: 0.12, 6400 train 99.97%, 1600 test 91.50%

From the experimentation results, the use of the learning rate scheduler proved to produce similar or better performance across models. For the model RegNet_Y_800MF_Weights.IMAGENET1K_V2, the highest test accuracy across all model experiments and model design parameters was achieved, reaching a test accuracy of 93.88%.

Hence, a learning rate scheduler will be used for further experimentation in the next steps and in the final model.

Step 5. Final Model Selection For Further Fine-tuning

As the model RegNet_Y_800MF_Weights.IMAGENET1K_V2 has achieved quite high test accuracy, future experimentation in the next steps in this project's method will only use this model. This will allow for quicker experimentation and to determine if this model can be further fine-tuned to achieve increased accuracy.

Step 6. Weight Decay Experimentation

As discussed in Step 2, weight decay is a regularisation technique that helps prevent overfitting. Several experiments were conducted with varying values of weight decay - 0.1, 0.01 (default choice in Step 2), 0.001 and 0.0001.

These experiments were conducted on the best performing model as determined in Step 4 - RegNet_Y_800MF_Weights.IMAGENET1K_V2.

Experimentation Results

Model	**Batch Size**	**Epochs**	**Weight Decay**	**Results**
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.0001	ep 30, loss: 0.10, 6400 train 99.98%, 1600 test 92.00%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.001	ep 30, loss: 0.06, 6400 train 100.00%, 1600 test 90.75%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.1	ep 30, loss: 0.11, 6400 train 99.98%, 1600 test 91.88%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.01	ep 20, loss: 0.16, 6400 train 100.00%, 1600 test 93.88%

Based on these results, the weight decay value of 0.01 produced the best results on the test set, achieving a test accuracy of 93.88%. Hence, the weight decay value of 0.01 will be used for the final model design.

Step 7. Optimizer Function Experimentation

In previous steps, all experiments with pre-trained models have used the Adam optimizer function variant - AdamW. However, stochastic gradient descent is another popular optimizer function.

Experiments were conducted using stochastic gradient descent on the best performing model as determined in Step 5 - RegNet_Y_800MF_Weights.IMAGENET1K_V2, to assess whether SGD produces better performing models in comparison to AdamW.

The SGD optimizer function has a parameter - momentum - that requires tuning. The selected momentum values for experimentation were: 0.5, 0.9, 0.95 and 0.99.

SGD also requires a learning rate value. The learning rate selected with this optimizer function was 0.001, as it usually requires a higher learning rate value in comparison to AdamW where the value of 0.0001 was used.

SGD also requires a weight decay value. The weight decay value of 0.01 was selected based on results from Step 5.

Experimentation Results

Model	**Batch Size**	**Epochs**	**Momentum**	**Results**
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.5	ep 30, loss: 21.63, 6400 train 94.00%, 1600 test 86.81%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.9	ep 30, loss: 2.05, 6400 train 99.86%, 1600 test 90.50%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.95	ep 30, loss: 1.34, 6400 train 99.97%, 1600 test 91.62%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	30	0.99	ep 30, loss: 1.13, 6400 train 99.97%, 1600 test 90.56%

Based on these results, the momentum value 0.95 produced the best results on the test set, achieving a test accuracy of 91.62%.

However, in comparison to experiments with the AdamW optimizer instead, SGD does not outperform AdamW, where the highest test accuracy achieved was 93.88%.

Hence, for the final model design, the AdamW optimizer function will be used.

Step 8. Learning Rate Experimentation

As discussed in Step 4, selecting a good learning rate value is important for a model to properly converge and lead to better model generalisation. Overfitting or underfitting of a model is common with a poor choice of learning rate.

In previous steps, all experiments with pre-trained models have used the learning rate value of 0.0001. Although using a learning rate scheduler has been experimented with in Step 4 which does adjust the learning rate throughout training, the initial learning rate value has stayed constant at 0.0001.

Experiments were conducted using other learning rate values on the best performing model as determined in Step 5 - RegNet_Y_800MF_Weights.IMAGENET1K_V2, to assess whether other learning rates produce better performing models in comparison to the learning rate of 0.0001.

The selected learning rate values for experimentation were: 0.01, 0.001, 0.0001 and 0.00001. The Adam optimizer was used as determined in Step 7. Values for weight decay remained at 0.01 as per results in Step 6 and all other parameters as defined in Step 2.

Experimentation Results

Model	**Batch Size**	**Epochs**	**Learning Rate**	**Results**
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.01	ep 20, loss: 0.91, 6400 train 99.83%, 1600 test 79.00%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.001	ep 20, loss: 0.06, 6400 train 99.98%, 1600 test 91.62%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.0001	ep 20, loss: 0.16, 6400 train 100.00%, 1600 test 93.88%
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.00001	ep 30, loss: 3.65, 6400 train 99.31%, 1600 test 89.50%

Based on these results, the learning rate value of 0.0001 produced the best results on the test set, achieving a test accuracy of 93.88%.

Hence, for the final model design, the learning rate value of 0.0001 will be used.

Step 9. Validation Split Experimentation

As mentioned in Step 2, having a validation set can be used to help with overfitting. It can be observed if a model is memorising or generalising the training data by observing the values for the training set accuracy and the test set accuracy.

In previous steps, a training / validation split of 0.8 has been used. This value was selected to provide a good balance between having enough data for training and having enough data in the validation set to more accurately estimate a model's performance, as described in Step 2.

An additional experiment was conducted using a value of 0.9 as the training / validation split on the best performing model as determined in Step 5 - RegNet_Y_800MF_Weights.IMAGENET1K_V2, to assess whether using more training data can yield better model performance and generalisation. Values below 0.8 were not considered to ensure there's enough training data.

Experimentation Results

Model	Batch Size	Epochs	Training / Validation Split	Results
RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.8	ep 1, loss: 94.76, 6400 train 72.20%, 1600 test 89.94% ep 2, loss: 21.77, 6400 train 93.39%, 1600 test 91.94% ep 3, loss: 8.18, 6400 train 98.14%, 1600 test 92.50% ep 4, loss: 3.74, 6400 train 99.17%, 1600 test 93.19% ep 5, loss: 1.98, 6400 train 99.61%, 1600 test 93.38% ep 6, loss: 1.14, 6400 train 99.88%, 1600 test 93.50% ep 7, loss: 0.82, 6400 train 99.91%, 1600 test 92.81% ep 8, loss: 0.67, 6400 train 99.97%, 1600 test 93.12% ep 9, loss: 0.41, 6400 train 100.00%, 1600 test 93.44% ep 10, loss: 0.48, 6400 train 99.95%, 1600 test 93.12% ep 11, loss: 0.39, 6400 train 99.97%, 1600 test 93.31% ep 12, loss: 0.35, 6400 train 99.97%, 1600 test 93.12% ep 13, loss: 0.28, 6400 train 99.95%, 1600 test 93.31% ep 14, loss: 0.28, 6400 train 99.98%, 1600 test 92.81% ep 15, loss: 0.29, 6400 train 99.95%, 1600 test 93.44% ep 16, loss: 0.20, 6400 train 99.98%, 1600 test 93.81% ep 17, loss: 0.19, 6400 train 100.00%, 1600 test 93.19% ep 18, loss: 0.20, 6400 train 100.00%, 1600 test 93.38% ep 19, loss: 0.17, 6400 train 99.98%, 1600 test 94.12% ep 20, loss: 0.16, 6400 train 100.00%, 1600 test 93.88%

RegNet_Y_800MF_Weights.IMAGENET1K_V2	64	20	0.9	ep 1, loss: 97.48, 7200 train 73.79%, 800 test 90.75% ep 2, loss: 24.11, 7200 train 93.38%, 800 test 92.00% ep 3, loss: 9.51, 7200 train 97.81%, 800 test 92.62% ep 4, loss: 4.33, 7200 train 99.22%, 800 test 93.00% ep 5, loss: 2.40, 7200 train 99.65%, 800 test 91.62% ep 6, loss: 1.35, 7200 train 99.85%, 800 test 91.75% ep 7, loss: 0.91, 7200 train 99.88%, 800 test 93.25% ep 8, loss: 0.85, 7200 train 99.83%, 800 test 92.25% ep 9, loss: 1.11, 7200 train 99.78%, 800 test 91.88% ep 10, loss: 0.82, 7200 train 99.88%, 800 test 92.88% ep 11, loss: 0.58, 7200 train 99.89%, 800 test 91.75% ep 12, loss: 0.55, 7200 train 99.94%, 800 test 92.50% ep 13, loss: 0.50, 7200 train 99.90%, 800 test 92.50% ep 14, loss: 0.60, 7200 train 99.90%, 800 test 92.50% ep 15, loss: 0.47, 7200 train 99.92%, 800 test 92.12% ep 16, loss: 0.21, 7200 train 99.99%, 800 test 92.25% ep 17, loss: 0.22, 7200 train 99.96%, 800 test 91.75% ep 18, loss: 0.14, 7200 train 100.00%, 800 test 92.25% ep 19, loss: 0.13, 7200 train 99.99%, 800 test 92.25% ep 20, loss: 0.12, 7200 train 100.00%, 800 test 92.75%

By observing the train and test accuracies per epoch, some observations can be made about the model performance and generalisation.

Experiment with 0.8 training / validation split:
It seems that the trained model in this experiment has mild overfitting, but generalisation is good. Some overfitting can be observed as training accuracy is around 100% from epoch 16 onwards, while the test accuracy ranges from 93-94%. Some overfitting can be observed as the train loss keeps dropping and the accuracy saturates at 100%, while the test accuracy oscillates within a range and then dips slightly after epoch 19.

Experiment with 0.9 training / validation split:

It seems that the trained model in this experiment has high overfitting. Overfitting can be observed from epoch 11 onwards, where the train accuracy sits close to 100% while the test accuracy stalls around 91.7-92.8%. There is a persistent 7-8% gap with the decreasing train loss and flat test accuracy, both signalling common overfitting. Overfitting is likely due to having a small validation set.

Based on these results, training / validation split values above 0.9 were not explored, as even at the value of 0.9, it seems the validation set is too small.

Furthermore, the model trained with a 0.8 training / validation split seems to be better at generalising and hence this value will be used for the final model design.

Step 10. Final Model Design and Parameter Selection

Final Selected Model and Parameter Values

Pre-trained Classification Model: RegNet_Y_800MF_Weights.IMAGENET1K_V2
Loss function: Cross Entropy
Data transformation: Resizing & normalisation using calculated mean and standard deviation of actual dataset
Batch size: 64
Number of epochs: 20
Weight decay: 0.01
Learning rate: 0.0001
Learning rate schedular: CosineAnnealingLR
Optimizer function: AdamW
Training & validation split: 0.8

Final Model Results

Final accuracy results: 100.00% accuracy on training dataset, 93.88% accuracy on testing dataset

Conclusion

The aim of this project was to find and train the most accurate neural network model on the provided 8000 images of cats and categorise each image into its relevant category.

Through the experimentation method untaken, a model was selected and fine-tuned to produce high accuracy results on the dataset. The pre-trained pytorch image classification model - RegNet_Y_800MF - was selected. After fine-tuning and careful parameter selection, this model yielded 100% accuracy on the training dataset and 93.88% accuracy on a sample testing dataset.