Assessing Fracture Healing
with Artificial Intelligence:
Using Transfer Learning to Predict the
Radiographic Union Score for Tibial Fractures,
in the Radiography of High-Energy Trauma

Shen Zhou Hong   Goldsmiths, UoL

April 4th, 2023

Contents

Chapter 1

Implementation

1.1 K-Fold Evaluation

```python
def k_fold_dataset(ds: tf.data.Dataset, k: int = 10) -> list[tuple[tf.data.Dataset,
↪ tf.data.Dataset]]:
    # First shard the given dataset into k individual folds.
    list_of_folds: list[tf.data.Dataset] = []
    for i in range(k):
        fold: tf.data.Dataset = ds.shard(num_shards=k, index=i)
        list_of_folds.append(fold)

    # Next, generate a list of train and validation dataset tuples
    list_of_ds_pairs: list[tuple[tf.data.Dataset, tf.data.Dataset]] = []
    for i, holdout_fold in enumerate(list_of_folds):
        ds_valid: tf.data.Dataset = holdout_fold

        # Select every fold except holdout_fold as the training folds
        training_folds: list[tf.data.Dataset] = list_of_folds[:i] +
        ↪ list_of_folds[i+1:]

        # ds_train size is (k-1)/k of the original dataset
        ds_train: tf.data.Dataset = training_folds[0]
        for fold in training_folds[1:]:
            ds_train = ds_train.concatenate(fold)

        ds_pair: tuple[tf.data.Dataset, tf.data.Dataset] = (ds_train, ds_valid)
        list_of_ds_pairs.append(ds_pair)

    return list_of_ds_pairs
```

Listing 1: Sharding dataset for K-Fold Cross Validation (Github)

```python
def cross_validate(ModelClass: tf.keras.Model, ds: tf.data.Dataset, epochs: int = 50,
↪   batch_size: int = 128, k: int = 10) -> list[tf.keras.callbacks.History]:

    history_list: list[tf.keras.callbacks.History] = []
    train_valid_pairs: list[tf.data.Dataset] = k_fold_dataset(ds, k)

    for i, (ds_train, ds_valid) in enumerate(train_valid_pairs):

        tf.keras.backend.clear_session()
        model = ModelClass()
        model.compile(
            optimizer=tf.keras.optimizers.Adam(),
            loss=tf.keras.losses.BinaryCrossentropy(),
            metrics=metrics
        )
        history = model.fit(
            ds_train,
            validation_data=ds_valid,
            epochs=epochs,
            batch_size=batch_size,
        )
        history_list.append(history.history)

    return history_list
```

Listing 2: K-Fold Cross Validation (Github)

```python
def calculate_mean_metrics(kfold_metrics: list[dict[str, float]]) -> dict[str,
↪   list[float]]:
    # Initialise aggregate metrics with appropriate keys
    aggregate_metrics: dict[str, list[float]] = {}
    for fold in kfold_metrics:
        for metric in fold.keys():
            if metric not in aggregate_metrics:
                aggregate_metrics[metric] = []

    # Calculate the average metric per epoch for every fold
    number_of_folds: int = len(kfold_metrics)
    for metric in aggregate_metrics.keys():
        number_of_epochs: int = len(kfold_metrics[0][metric])
        for epoch in range(number_of_epochs):
            # A list of every value for that given metric in this epoch across folds
            values_per_epoch: list[float] = [x[metric][epoch] for x in kfold_metrics]
            mean_per_epoch  : float = sum(values_per_epoch) / number_of_folds
            aggregate_metrics[metric].append(mean_per_epoch)

    return aggregate_metrics
```

Listing 3: Calculating Mean Metrics from K-Fold Data (Github)

1.2 Establishing a Baseline

1.2.1 Shallow Convolutional Neural Network

```python
class LeNet1998(tf.keras.Model):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

        self.input_layer: tf.Tensor = layers.InputLayer(input_shape=(299, 299, 3))
        self.data_augmentation: tf.keras.Sequential = tf.keras.Sequential([
            layers.RandomFlip(seed=RNG_SEED),
        ])

        self.lenet1999: tf.keras.Model = tf.keras.Sequential([
            layers.Conv2D(6, kernel_size=5, strides=1,  activation='tanh',
            ↪  padding='same'),
            layers.AveragePooling2D(),
            layers.Conv2D(16, kernel_size=5, strides=1, activation='tanh',
            ↪  padding='valid'),
            layers.AveragePooling2D(),
        ])

        self.classifier: tf.keras.Sequential = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(1024, activation='relu'),
            layers.Dense(18, activation='sigmoid')
        ])

        self.model: tf.keras.Sequential = tf.keras.Sequential([
                self.input_layer,
                self.data_augmentation,
                self.lenet1999,
                self.classifier
        ])

    def call(self, inputs):
        return self.model(inputs)
```
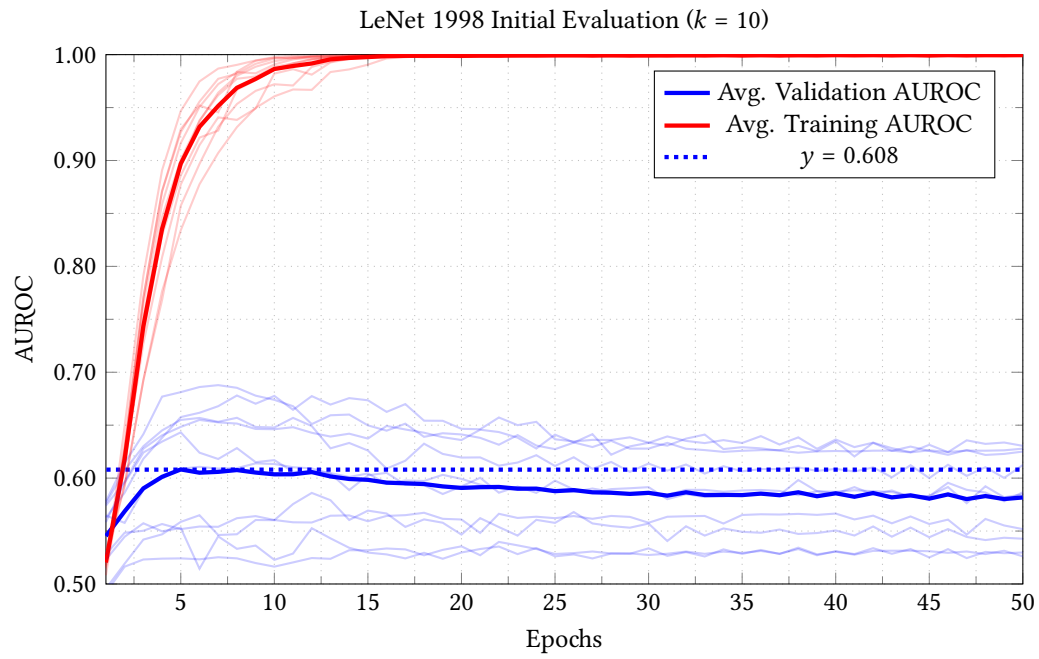
Listing 4: The LeNet 1998 Shallow CNN Model (Github)

Figure 1.1: Baseline shallow CNN based on the LeNet 1998 architecture

1.2.2 End-to-End Training with InceptionV3

```python
class TransferLearningModel(tf.keras.Model):
    def __init__(self, dropout_rate: float, **kwargs):
        super().__init__(**kwargs)

        self.input_layer: tf.Tensor = layers.InputLayer(input_shape=(299, 299, 3))
        self.data_augmentation: tf.keras.Sequential = tf.keras.Sequential([
            layers.RandomFlip(seed=RNG_SEED),
        ])

        self.inceptionv3: tf.keras.Model = tf.keras.applications.InceptionV3(
            include_top=False,
            weights='imagenet'
        )
        self.inceptionv3.trainable = False

        self.classifier: tf.keras.Sequential = tf.keras.Sequential([
            layers.GlobalMaxPooling2D(),
            layers.Dense(1024, activation='relu'),
            layers.Dropout(dropout_rate),
            layers.Dense( 512, activation='relu'),
            layers.Dropout(dropout_rate),
            layers.Dense( 256, activation='relu'),
            layers.Dropout(dropout_rate),
            layers.Dense(  18, activation='sigmoid')
        ])

        self.model: tf.keras.Sequential = tf.keras.Sequential([
            self.input_layer,
            self.data_augmentation,
            self.inceptionv3,
            self.classifier
        ])

    def call(self, inputs):
        return self.model(inputs)
```
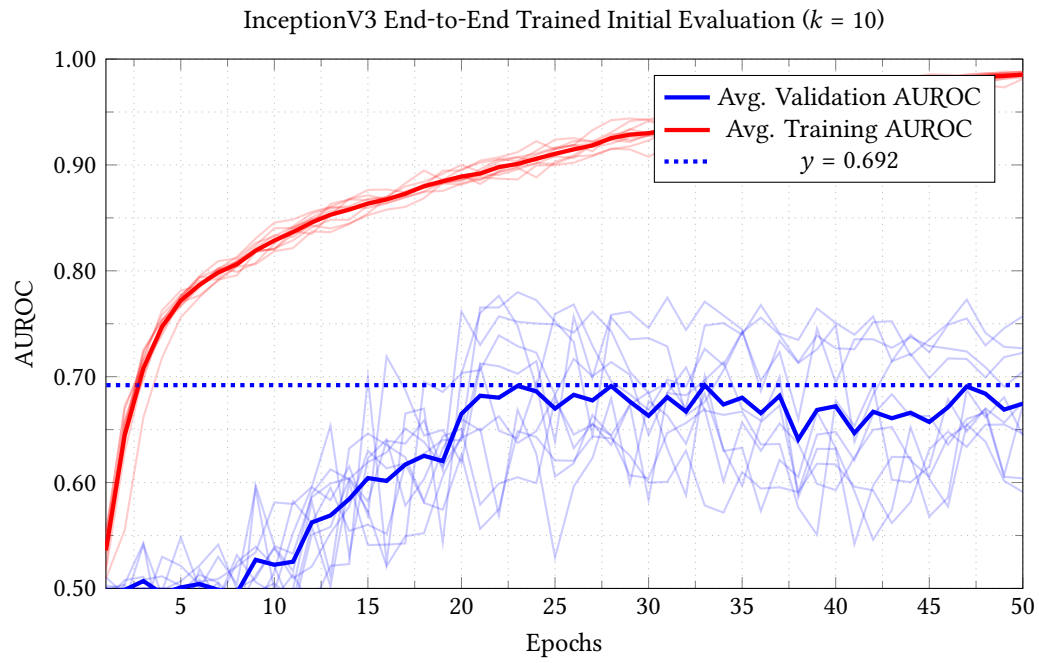
Listing 5: Model Class for InceptionV3 (Github)

InceptionV3 End-to-End Trained Initial Evaluation ($k = 10$)

Figure 1.2: InceptionV3 Model Trained on Study Data.

1.2.3 Baseline Metrics

1.3 InceptionV3 with Transfer Learning
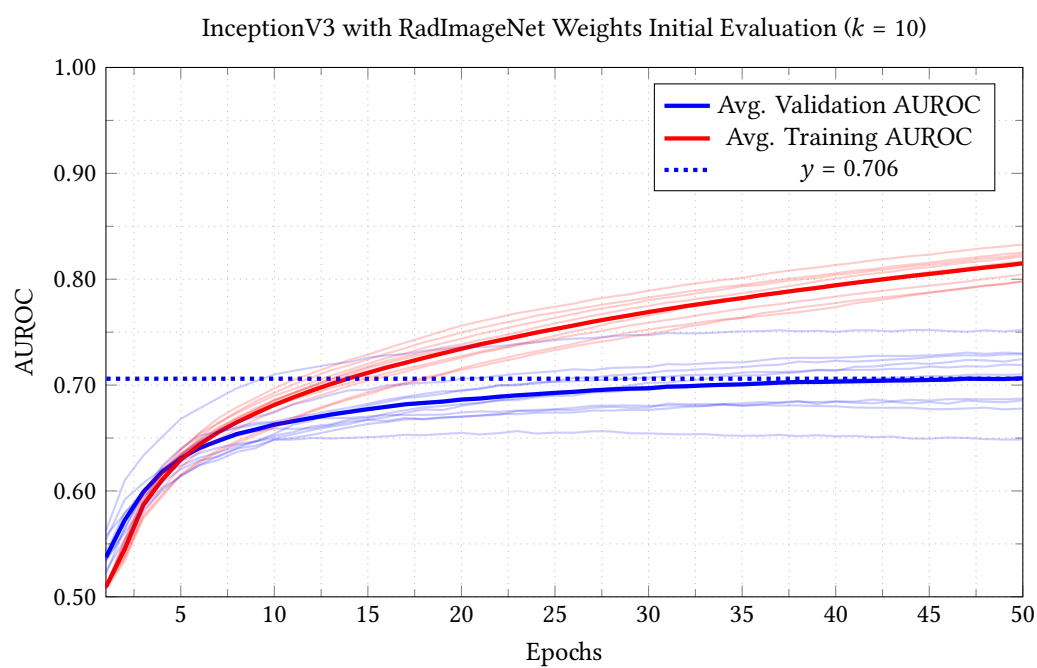
1.3.1 Base Model Trained on RadImageNet Dataset

Figure 1.3: InceptionV3 with RadImageNet Weights
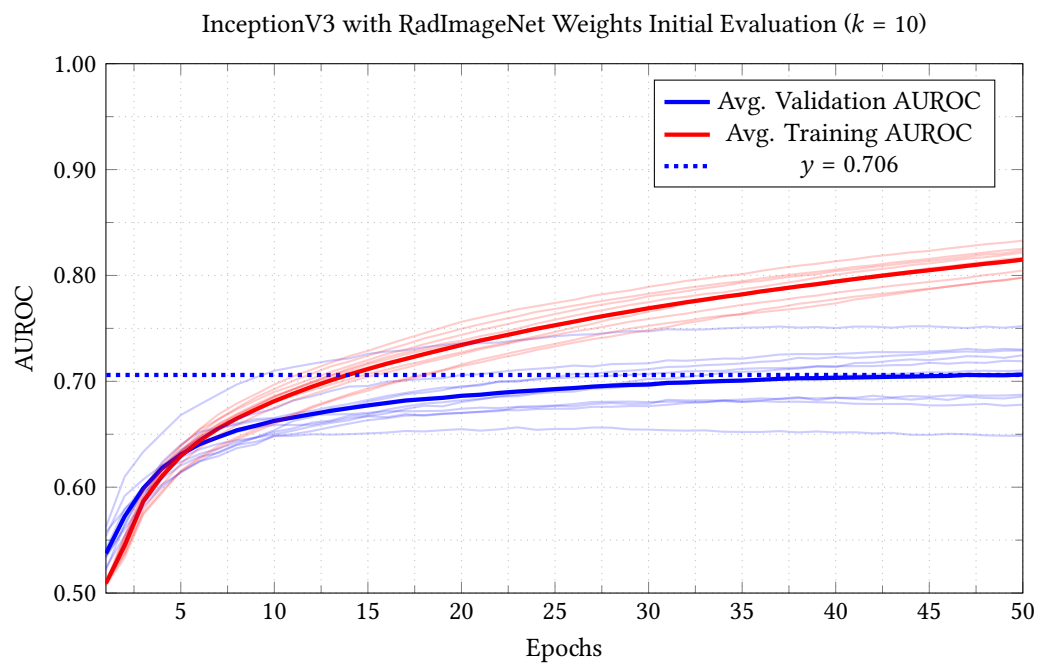
1.3.2 Base Model Trained on InceptionV3 Dataset

InceptionV3 with RadImageNet Weights Initial Evaluation ($k = 10$)

Figure 1.4: InceptionV3 with RadImageNet Weights

1.3.3 Comparison between RadImageNet and ImageNet

1.4 Hyperparameter Search

1.4.1 Hyperparameter Search Regime I

```python
def hyperparameter_search(trials: int, kfolds: int = 6, epochs: int = 20) ->
↪ list[dict[str, Union[int, float, list[tf.keras.callbacks.History]]]]:
    search_results: list[dict[str, any]] = []

    for trial in range(trials):
        # Randomly pick hyperparameter options
        rng = np.random.default_rng()
        batch_size : int   = rng.integers(16, 2048, endpoint=True)
        dropout_rate: float = rng.uniform(0.0, 0.5)

        # Conduct K-Fold cross-validation with given hyperparameters
        results: list[tf.keras.callbacks.History] = cross_validate(
            TransferLearningModel,
            ds_train_and_valid,
            k=kfolds
            epochs=epochs,
            batch_size=batch_size,
            model_kwargs={"dropout_rate": dropout_rate},
        )

        search_results.append({
            "batch_size"  : batch_size,
            "dropout_rate": dropout_rate,
            "history_list": k_fold_results
        })

    return search_results
```

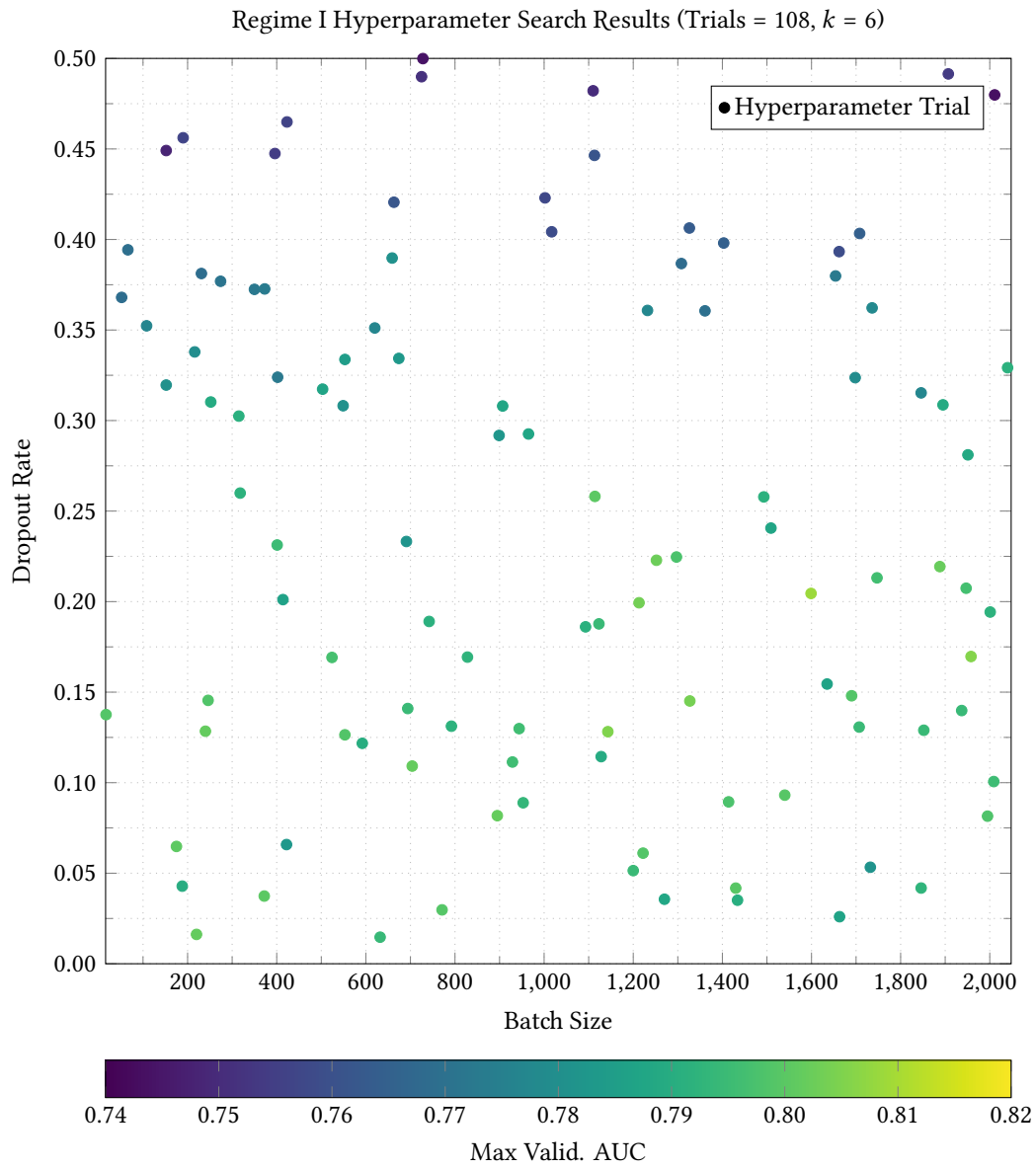Listing 6: Hyperparameter Search Regime I ([Github](#))

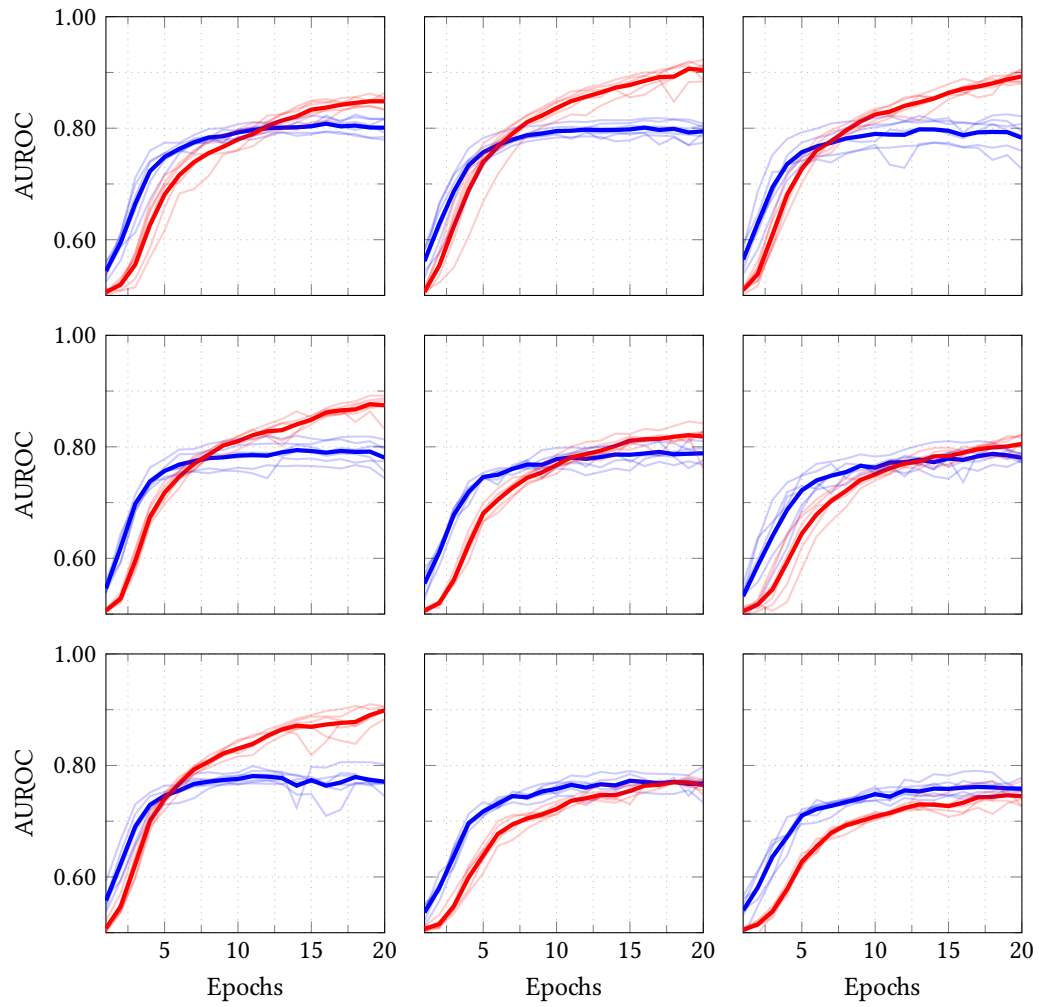Figure 1.5: Results for the Hyperparameter Search Regime I

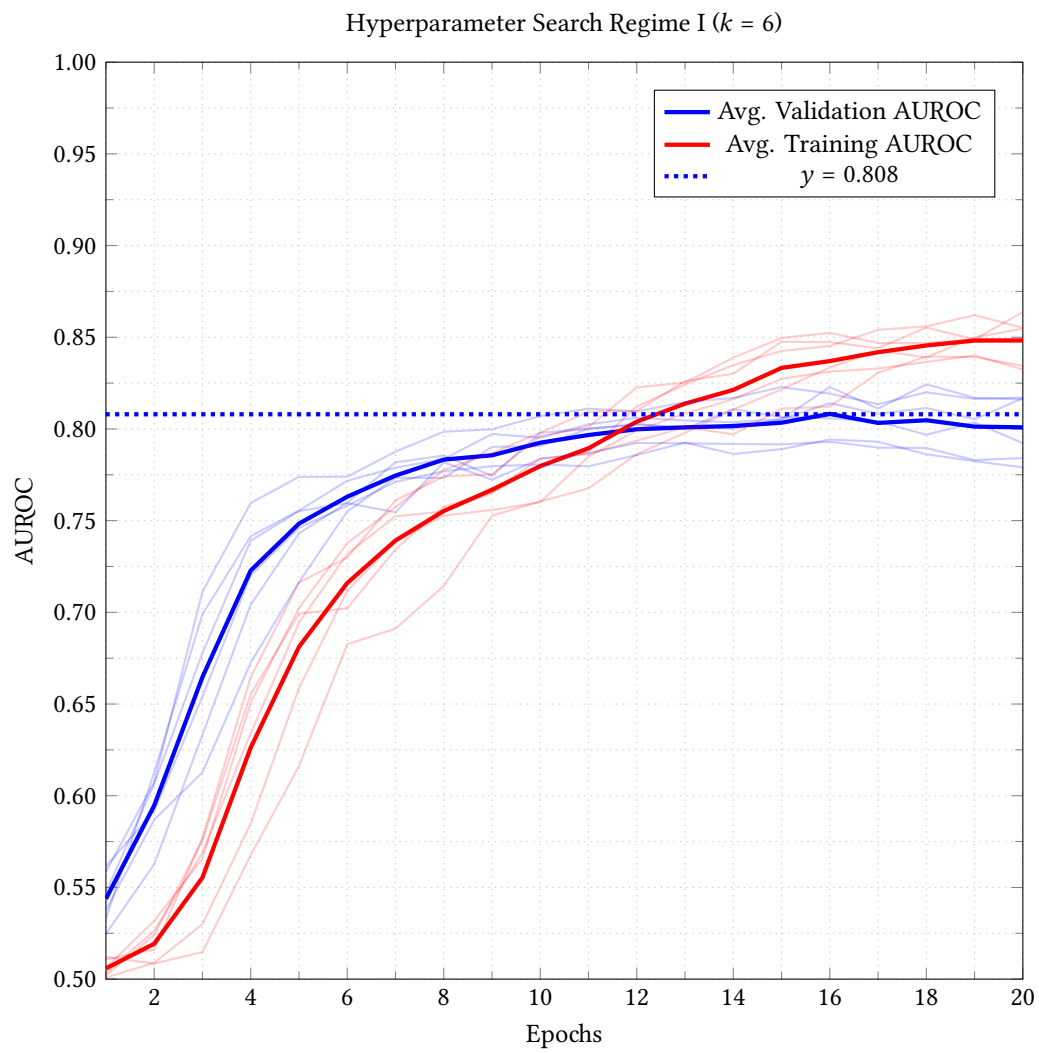Figure 1.6: Examples of model performance from hyperparameter regime I search.

Figure 1.7: Best performing model in Regime I

1.4.2 Hyperparameter Search Regime II

```python
def learning_rate_gridsearch(kfolds: int = 6) -> list[dict[str, Union[int, float,
↪ list[tf.keras.callbacks.History]]]]:
    # Grid i: 1.0 × 10⁻¹ ≤ learning_rate ≤ 1.0 × 10⁻⁵
    learning_rates: list = [1 * np.float_power(10, -exp) for exp in range(1, 5)]
    # Grid j: 1.0 × 10⁻¹ ≤ epsilon_rate ≤ 1.0 × 10⁻⁸
    epsilon_rates : list = [1 * np.float_power(10, -exp) for exp in range(1, 9)]

    search_results: list[dict[str, Union[int, float,
    ↪ list[tf.keras.callbacks.History]]]] = []
    for i, learning_rate in enumerate(learning_rates):
        for j, epsilon_rate in enumerate(epsilon_rates):
            # Conduct K-Fold Experiment
            k_fold_results: list[tf.keras.callbacks.History] = cross_validate(
                TransferLearningModel,
                ds_train_and_valid,
                k=kfolds,
                epochs=EPOCHS,
                batch_size=BATCH_SIZE,
                model_kwargs={"dropout_rate": DROPOUT_RATE}
                optimizer_kwargs={"learning_rate": learning_rate, "epsilon":
                ↪ epsilon_rate},
            )
            search_results.append({
                "learning_rate": learning_rate,
                "epsilon_rate" : epsilon_rate,
                "history_list" : k_fold_results
            })

    return search_results
```
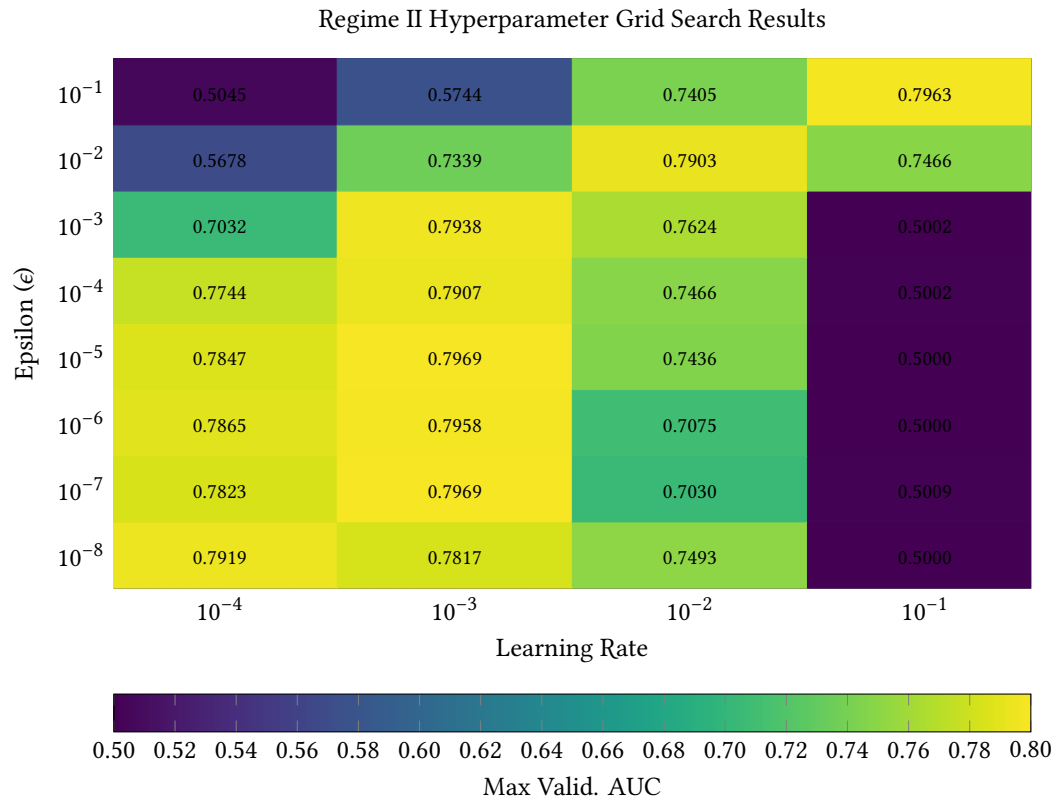
Listing 7: Hyperparameter Search Regime II (Github)

Regime II Hyperparameter Grid Search Results

	10^{-4}	10^{-3}	10^{-2}	10^{-1}
10^{-1}	0.5045	0.5744	0.7405	0.7963
10^{-2}	0.5678	0.7339	0.7903	0.7466
10^{-3}	0.7032	0.7938	0.7624	0.5002
10^{-4}	0.7744	0.7907	0.7466	0.5002
10^{-5}	0.7847	0.7969	0.7436	0.5000
10^{-6}	0.7865	0.7958	0.7075	0.5000
10^{-7}	0.7823	0.7969	0.7030	0.5009
10^{-8}	0.7919	0.7817	0.7493	0.5000

Epsilon (ϵ) — Learning Rate

Max Valid. AUC
0.50 0.52 0.54 0.56 0.58 0.60 0.62 0.64 0.66 0.68 0.70 0.72 0.74 0.76 0.78 0.80

Figure 1.8: Results for the Hyperparameter Search Regime II

1.4.3 Final Hyperparameters

1.5 Final Model Performance

Appendix A

Additional Materials

A.1 Project Code and Github Repository

All of the Python code used in this project (including experiment and analysis code) are available within the project Git repository, hosted on Github. The code is located within the `python/` directory of the repository root:

https://github.com/ShenZhouHong/radiography-ai-project/

A.1.1 Initial Evaluation Models

Jupyter notebooks used to run the initial evaluations of LeNet 1998, InceptionV3 with end-to-end training, and initial transfer learning models:

https://github.com/ShenZhouHong/radiography-ai-project/tree/master/python/initial-evaluation

A.1.2 Hyperparameter Search Code

Jupyter notebooks used to perform the hyperparameter search regime.

https://github.com/ShenZhouHong/radiography-ai-project/tree/master/python/hyperparam-search

A.1.3 Analysis Notebooks

Jupyter notebooks used to analyse the raw data, process for insights and visualisations, and output CSV files:

https://github.com/ShenZhouHong/radiography-ai-project/tree/master/python/analysis