Travel Product Classification

Pipeline Report

Areeya Khaowaiyakul

Executive Summary

In the submitted model, I built a Keras neural network to predict whether a customer will purchase a travel product (ProdTaken) from a dataset of 3,208 customer records.

I first prepared the data by cleaning typos and outliers, then analyzed the data to understand patterns - discovering that Passport ownership is the strongest predictor (70% purchase rate vs 38% without). I applied random oversampling to fix class imbalance (81/19 to 50/50).

For feature extraction, I one-hot encoded 6 categorical columns (producing 28 features) and standardized all values. I chose one-hot over label encoding because the categories have no ordinal relationship.

I initially trained the baseline architecture from class (32->8->4->1) which achieved 88% accuracy, but wanted better. After experimenting with wider networks, BatchNormalization, and learning rate scheduling, the improved model (256->128->1) achieved significantly better results.

The final model achieved 96.45% accuracy and 0.99 AUC on the held-out test set - an improvement of ~8 percentage points over the baseline.

Pipeline

1. Data Preparation

The raw dataset had 3,208 rows and 19 columns describing customer demographics, sales pitch details, and whether the customer purchased the travel product. I found and fixed 3 data quality issues:

- Gender typo: 121 rows had "Fe Male" instead of "Female" - merged into "Female"
- DurationOfPitch outlier: 1 row had 127 min (next highest: 36) - replaced with median (14.0)
- NumberOfTrips outlier: 3 rows had values 20, 21, 22 (next highest: 8) - replaced with median (3.0)

The original target was heavily imbalanced - ProdTaken=0 had 2,589 rows (80.7%) while ProdTaken=1 had only 619 rows (19.3%). I applied random oversampling on the minority class to balance it to 50/50 (5,178 total rows), then shuffled the data to prevent ordering bias.

The balanced data was split into 3 separate files before any training:

Split	Rows	Percentage
Training	3,106	60%
Validation	777	15%
Testing	1,295	25%
Train + Val	3,883	75%

Figure 1: Data overview - balanced target, 60/15/25 split, age & gender distributions

To conclude: data was cleaned of typos and outliers, balanced via oversampling, and split into 3 separate files to prevent data leakage.

2. Analysis

I performed exploratory data analysis to understand what drives a customer to purchase. Key observations:

- Passport has the strongest correlation with ProdTaken (+0.313). Customers with a passport have a 69.7% purchase rate vs only 37.6% without - nearly double.
- NumberOfFollowups (+0.131) and CityTier (+0.128) are positively correlated - more follow-ups and higher-tier cities lead to more purchases.
- Age (-0.183) and MonthlyIncome (-0.180) are negatively correlated - younger customers with lower income are more likely to buy.
- ProductPitched matters: Basic products have the highest purchase rate (63.9%), while King products have the lowest (21.1%).

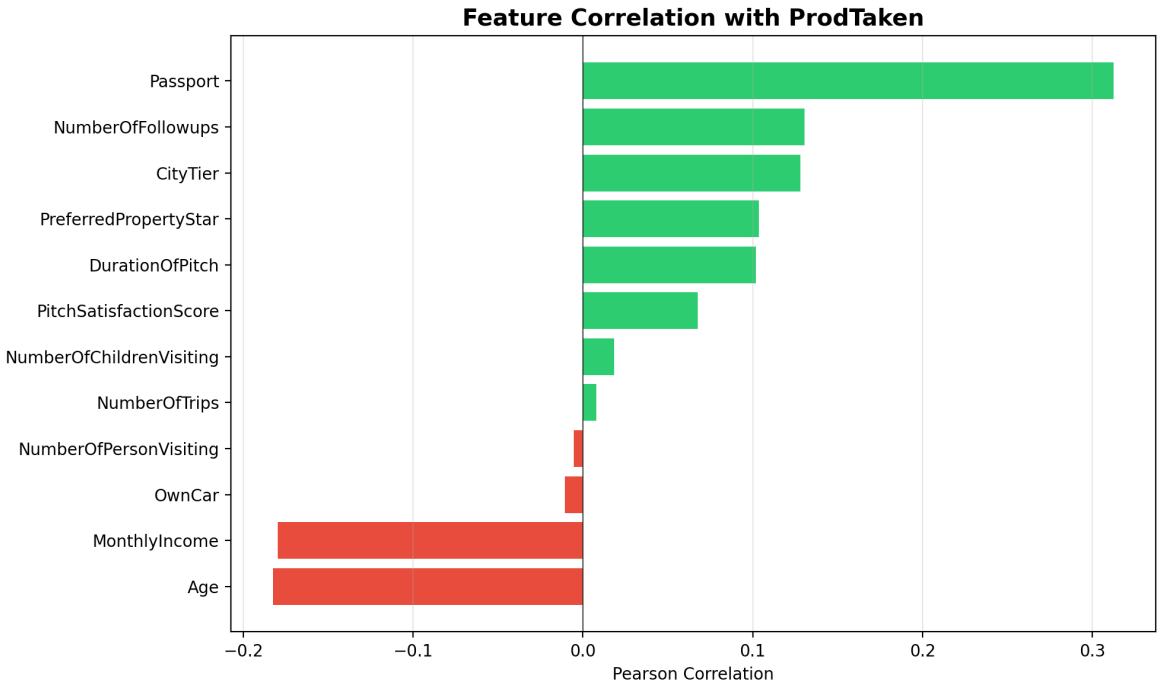

Figure 2: Feature correlation with ProdTaken - Passport is the strongest positive predictor

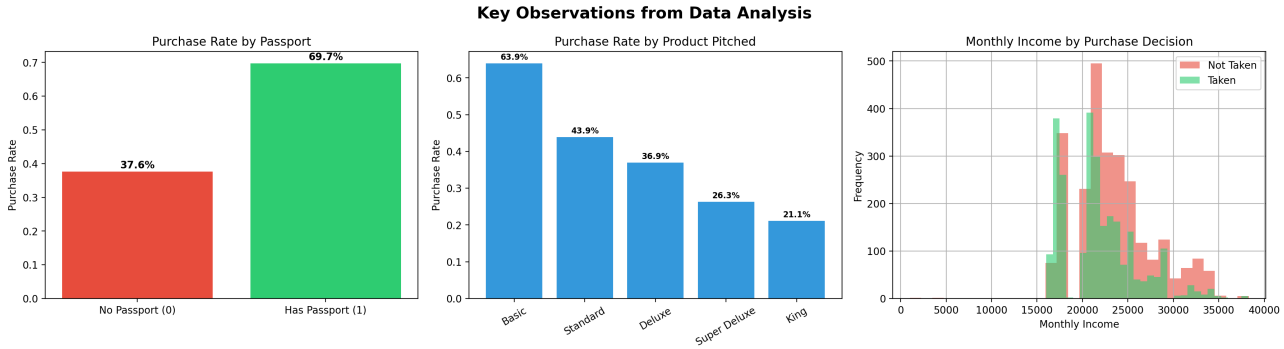

Figure 3: Passport doubles purchase rate, Basic products sell best, lower income customers buy more

To conclude: Passport ownership, product type, follow-ups, and city tier are the strongest predictors. Age and income are negatively correlated.

3. Feature Extraction

The dataset has 18 features: 12 numeric and 6 categorical (TypeofContact, Occupation, Gender, ProductPitched, MaritalStatus, Designation). Feature engineering decisions:

- One-hot encoding (not label encoding) for categoricals - because categories like "Basic", "Deluxe", "King" have no natural ordinal order. Label encoding would imply King > Deluxe > Basic, which is misleading.
- drop_first=True - to avoid the dummy variable trap (multicollinearity). For example, Gender only needs 1 column (Male=1), since Female is implied when Male=0.
- StandardScaler - neural networks are sensitive to feature scale. Without scaling, features with large ranges (e.g., MonthlyIncome: 5,000-40,000) would dominate features with small ranges (e.g., Passport: 0-1).
- All features kept - no features were dropped because the correlation analysis showed all features have some predictive value. The neural network can learn to ignore irrelevant features through its weights.

This produced 28 features from the original 18 (12 numeric stayed the same, 6 categoricals expanded to 16 dummy columns with drop_first).

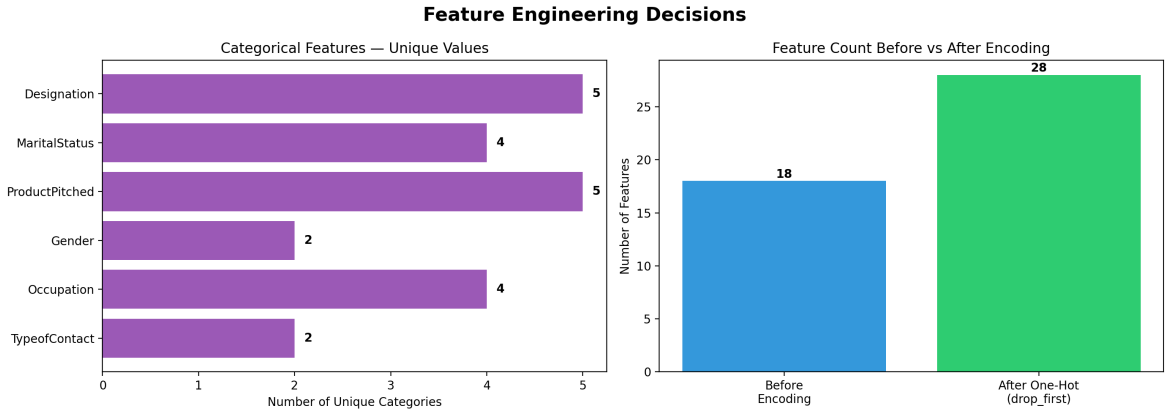

Figure 4: 6 categorical columns expanding 18 features to 28 after one-hot encoding

To conclude: one-hot encoding was chosen because categorical features have no ordinal relationship. drop_first prevents multicollinearity, and StandardScaler ensures all features contribute equally.

4. Building Model

Baseline Model (as taught in class)

Architecture: Dense(32, relu) -> Dropout(0.3) -> Dense(8, tanh) -> Dense(4, relu) -> Dropout(0.3) -> Dense(1, sigmoid)

Optimizer: Adam (lr=0.001) | Loss: Binary Cross-Entropy | Batch size: 16 | Early stopping: patience=5

Ran for 62 epochs. Result: 88.11% accuracy, 0.937 AUC

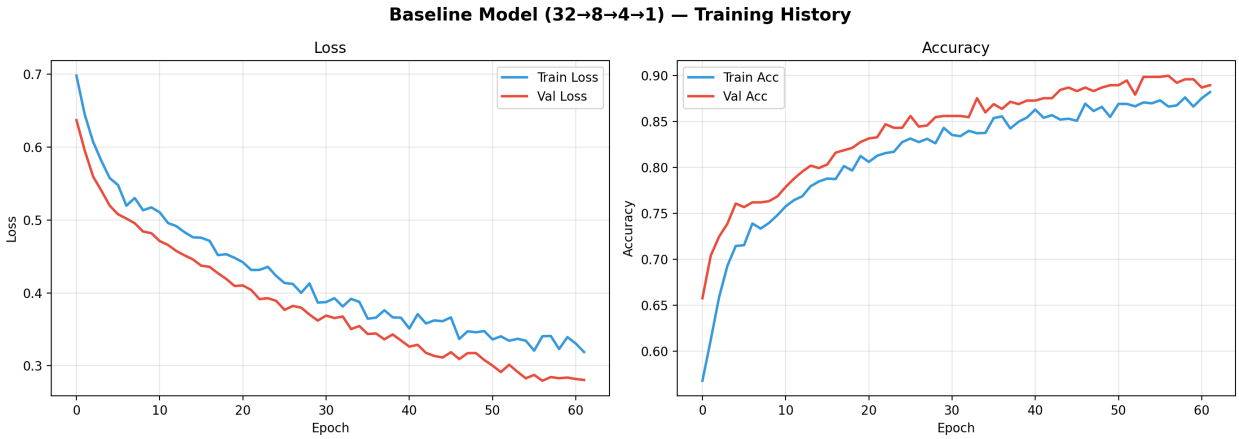

Figure 5: Baseline training history

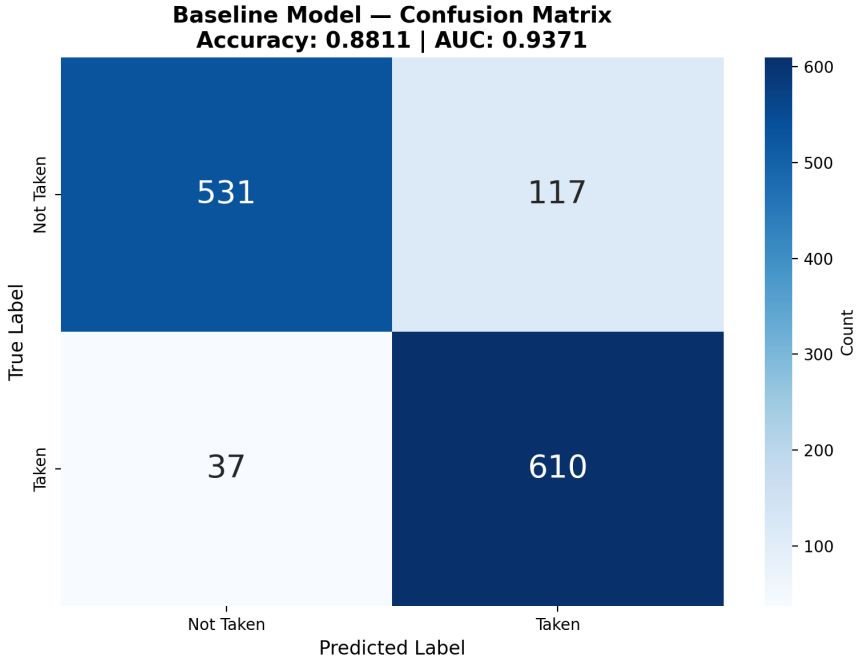

Figure 6: Baseline confusion matrix - 88.11% accuracy

The baseline was underfitting - not enough capacity. I experimented with 4 architectures:

Experiment	Architecture	Accuracy
Baseline (class)	32->8->4->1	88.11%
Exp 1: Wider	64->32->16 + BatchNorm	94.21%
Exp 2: Deep+ReduceLR	128->64->32->16 + BN	95.06%
Exp 3: LeakyReLU	128->64->32->16	93.59%
Exp 4: Wide shallow	256->128 + BatchNorm	96.14%

Improved Model (Experiment 4 - winner)

Architecture: Dense(256, relu) -> BatchNorm -> Dropout(0.4) -> Dense(128, relu) -> BatchNorm -> Dropout(0.3) -> Dense(1, sigmoid)

What I changed and why:

- Wider layers (256->128): more capacity to capture 28 feature interactions
- BatchNormalization: stabilizes training, prevents internal covariate shift
- Lower learning rate (0.0005): finer convergence, no overshooting
- ReduceLROnPlateau (factor=0.5, patience=7): LR halved 4 times during training
- Increased patience (20 instead of 5): baseline stopped too early
- Larger batch size (64): smoother gradient estimates

Ran for ~139 epochs. Result: 96.45% accuracy, 0.99 AUC

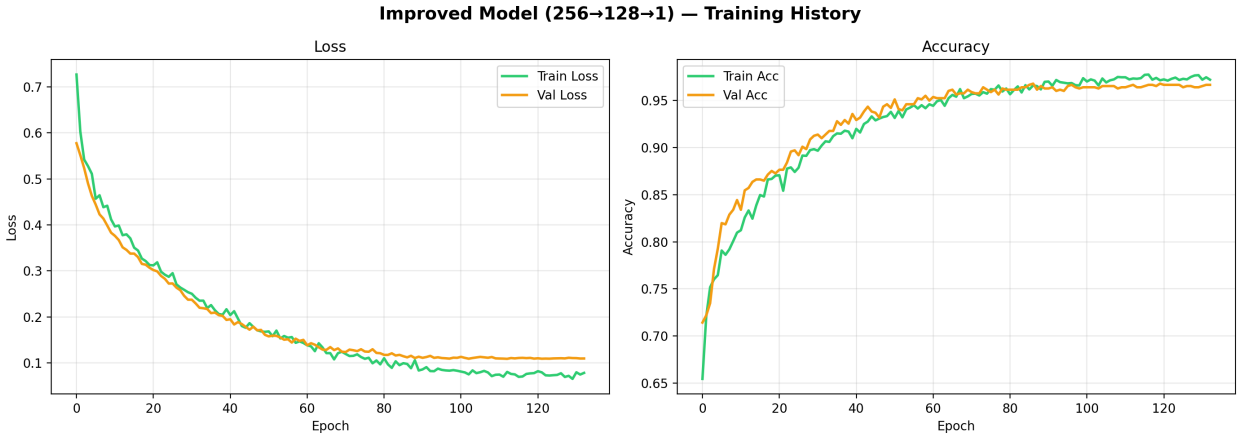

Figure 7: Improved model training

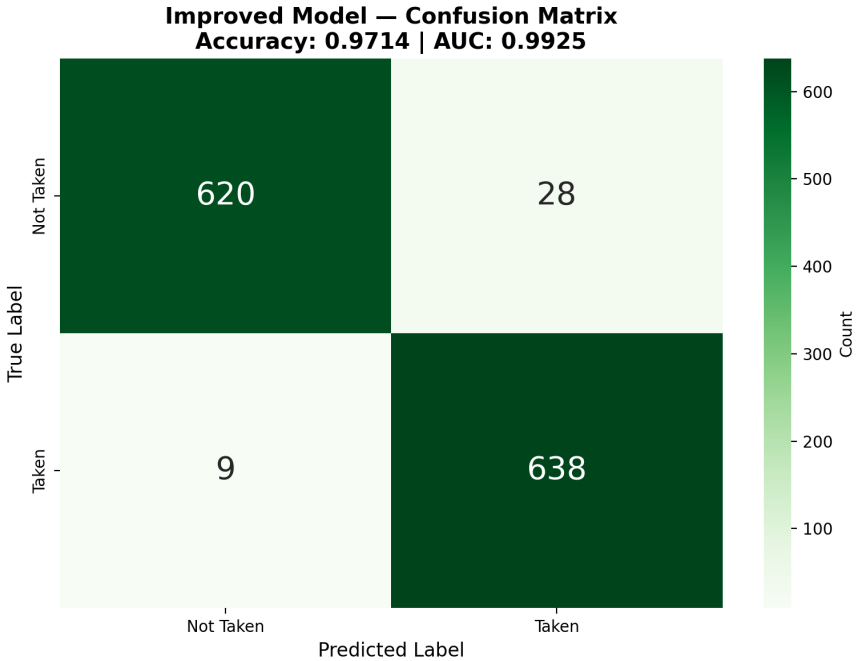

Figure 8: Improved confusion matrix - 96.45% accuracy

To conclude: the wider architecture with BatchNormalization and LR scheduling improved accuracy by ~8 percentage points (88% to 96%).

5. Evaluation Results

Both models were evaluated on the same held-out test set (1,295 samples, never seen during training):

Metric	Baseline (32->8->4->1)	Improved (256->128->1)
Accuracy	88.11%	96.45%
AUC	0.937	0.991
Precision (Not Taken)	0.93	0.98
Recall (Not Taken)	0.82	0.94
Precision (Taken)	0.84	0.95
Recall (Taken)	0.94	0.98
F1 (macro)	0.88	0.97

The improved model outperforms the baseline across every metric. False negatives dropped from 37 to 9.

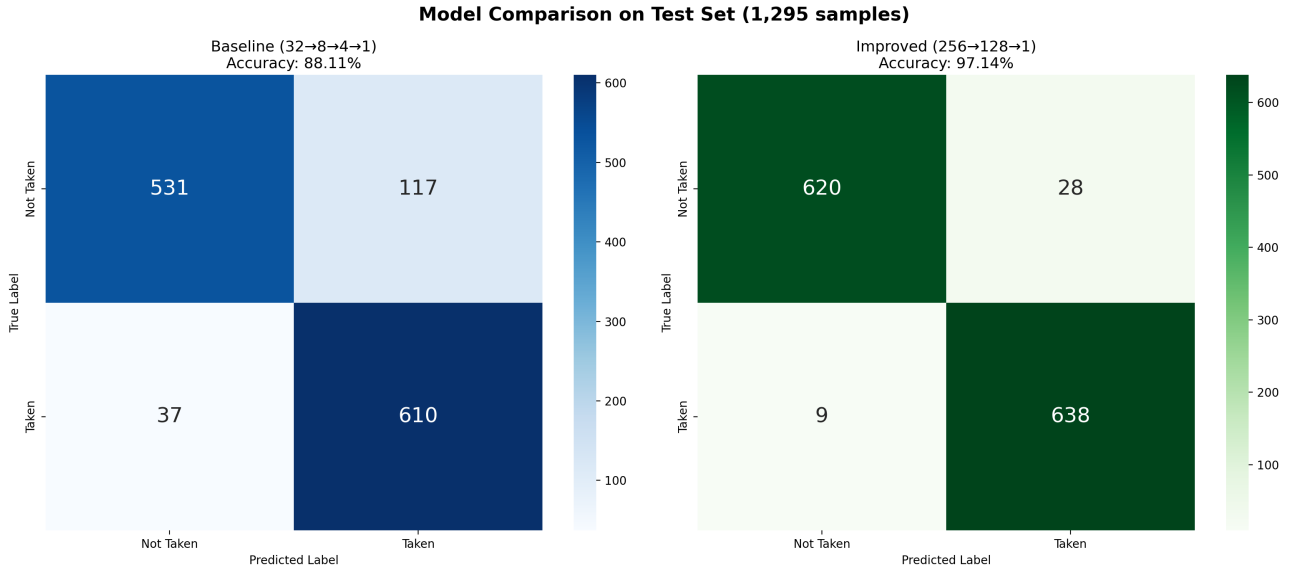

Figure 9: Side-by-side confusion matrices

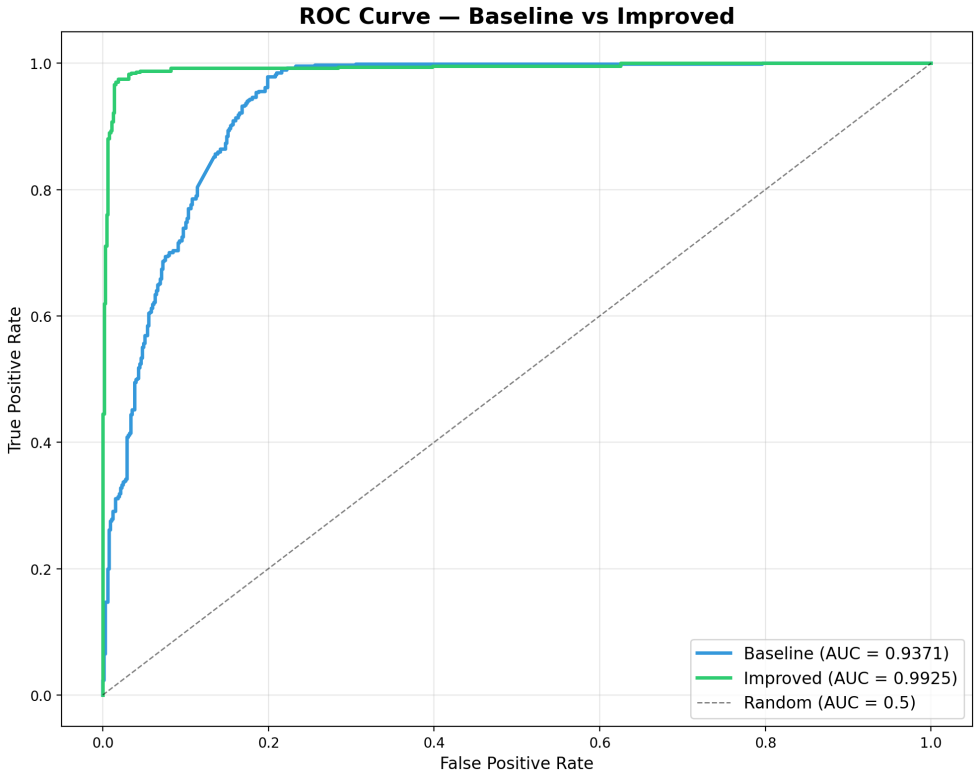

Figure 10: ROC curves - improved model AUC = 0.993

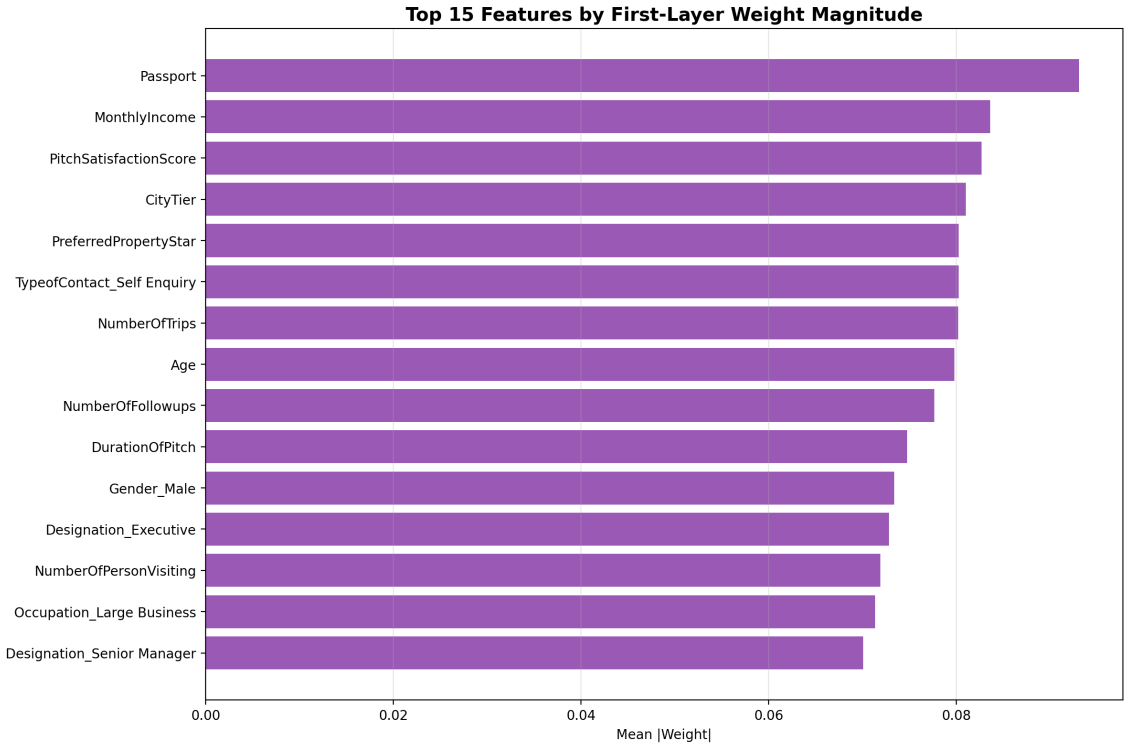

Figure 11: Top 15 features by weight magnitude

The feature importance from the trained model aligns with the correlation analysis from Stage 2: Passport, MonthlyIncome, and PitchSatisfactionScore are the most influential features.

To conclude: the final improved model (256->128->1 with BatchNormalization and ReduceLROnPlateau) achieved 96.45% test accuracy, 0.99 AUC, and 0.94-0.98 recall across both classes, significantly outperforming the baseline across all metrics.